НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ННК «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Кафедра системного проектування
(повна назва кафедри)

«На правах рукопису»
УДК _004.852

«До захисту допущено»

Завідувач кафедри
_____ _А.І. Петренко
(підпис) (ініціали, прізвище)

“___”_____20__ р.

Магістерська дисертація

зі спеціальності (спеціалізації)__8.05010103 Системне проектування_____
(код і назва спеціальності)

на тему: Семантичне оркестрування REST-сервісів

Виконав: студент _6_ курсу, групи ДА-62м
(шифр групи)

_____Магас Валентин Васильович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник _____ст.в. к.т.н. Булах Б.В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Розроблення стартап-проекту
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Інститут/факультет ННК «Інститут прикладного системного аналізу»
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною (освітньо-науковою) програмою

Спеціальність (спеціалізація) 8.05010103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І. Петренко
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Магасу Валентину Васильовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Семантичне оркестрування REST-сервісів»
науковий керівник дисертації _____Булах Б.В., к.т.н._____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Строк подання студентом дисертації

3. Об'єкт дослідження __Оркестрування сервісів_____

4. Предмет дослідження Підходи до семантичного оркестрування REST-сервісів_

5. Перелік завдань, які потрібно розробити провести огляд існуючих методів та технологій для створення опису REST-сервісів та подальшого оркестрування на їх основі, розробити математичну модель підходу до компонування сервісів, провести моделювання та аналіз роботи запропонованого рішення, оформити

роботу на основі отриманих результатів_____

6. Орієнтовний перелік публікацій Магас В.В. Компонування семантичних REST-сервісів з використанням Neo4J/ В.В.Магас. // Міжнародний науковий журнал "Інтернаука". – 2018. – №8. – С. 56–69._____

7. Консультанти розділів дисертації[1*]

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Реалізація стартап-проекту			

8. Дата видачі завдання _01.02.2018_

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.02.2018	
2	Збір інформації та аналіз літератури	15.02.2018	
3	Проведення огляду існуючих методів побудови мультиагентних систем	28.02.2018	
4	Формалізація задачі розробки методу прийняття колективного рішення в мультиагентних системах	11.03.2018	
5	Реалізація поставленої задачі	13.04.2018	
6	Аналіз та порівняння результатів моделювання	25.04.2018	
7	Оформлення дипломної роботи	30.04.2018	
8	Отримання допуску до захисту та подача роботи в ДЕК	09.05.2018	

Студент _____ ____Магас В.В._____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Булах Б.В._____
(підпис) (ініціали, прізвище)

РЕФЕРАТ

НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ

виконану на тему: Семантичне оркестрування REST-сервісів

студентом: Магасом Валентином Васильовичем

Робота виконана на 99 сторінках, містить 29 ілюстрацій, 27 таблиць. При підготовці використовувалась література з 21 джерела.

Актуальність теми

На відміну від сервісів на основі WSDL / SOAP, REST-сервіси не мають загальноприйнятого і використовуваного опису сервісів, оскільки він нарощує зв'язок між клієнтами і серверами, що ускладнює еволюцію сервісу. На практиці REST сервіси описуються через неофіційні, ad-hoc і напівструктуровані описи, часто написані природною мовою, що погіршує взаємодію. Основні підходи з опису REST сервісів в даний час пропонують слідувати операційно орієнтованому підходу з незрозумілими перевагами як для розробників так і споживачів. Як результат, тема набуває значної актуальності в контексті оркестрування REST-сервісів на основі відповідних описів.

Мета та задачі дослідження

Метою даної роботи є дослідження існуючих підходів до опису та компонування REST-сервісів та розробка власного рішення, яке б більшою мірою підходило для архітектурного стилю REST та надавало можливість автоматизованого компонування серівсів, грунтуючись на семантичних описах зрозумілих для машинних клієнтів.

Рішення поставлених завдань та досягнуті результати

Для вирішення поставлених завдань, у роботі пропонується метамодель, що передбачає два рівні взаємодії - семантичний та рівень активності. Семантику до сервісів було додано шляхом розширення словника Schema.org та використання нових елементів у JSON. За результатами згенерованих описів було побудовано граф, який в свою чергу використовується для знаходження

шляхів компонування сервісів. Запропоноване рішення було протестовано на таких веб-сервісах, як Spotify , Songkick і Uber. Результати роботи представлені цілою низкою діаграм, таблиць, блок-схем з детальним описом, та поясненнями.

Об’єкт досліджень

Семантичне оркестрування сервісів.

Предмет досліджень

Підходи до створення семантичних описів та шляхи оркестрування сервісів на їх основі.

Методи досліджень

Для вирішення проблеми в даній роботі використовуються методи аналізу і синтезу, системного аналізу, порівняння, логічного узагальнення результатів.

Наукова новизна

Наукова новизна роботи полягає у створенні нової моделі для вирішення задач опису та компонування REST-сервісів. З подальшою реалізацію пропонованих ідей та аналізом результатів.

Практичне значення одержаних результатів

Отримані результати являються новим кроком на шляху впровадження семантики в REST-сервіси. Продемонстрували можливість використання графових баз даних для цих цілей. Відкрили широке поле для подальших досліджень та вдосконалень.

Апробації результатів дисертації

Результати досліджень оприлюднені у міжнародному науковому журналі «Інтернаука», випуск №8 2018 року

Публікації

Магас В.В. Компонування семантичних REST-сервісів з використанням Neo4J/ В.В.Магас. // Міжнародний науковий журнал "Інтернаука". – 2018. – №8. – С. 56–69.

Ключові слова

REST-сервіси, JSON, графова база даних, семантичне оркестрування, онтологія, опис сервісів.

ABSTRACT

ON MASTER'S THESIS

on topic: Semantic orchestration of REST services

student: Valentyn V. Mahas

Work carried out on 99 pages containing 29 figures, 27 tables. The paper was written with references to 21 different sources.

Topicality

In contrast to services based on WSDL / SOAP, REST services do not have a commonly used description of services, as it increases coupling between clients and servers, which complicates the evolution of the service. In practice, REST services are described through unofficial, ad-hoc and semi-structured descriptions, often written in natural language, which leads to interaction problems. The basic approaches of describing REST services are following operationally oriented approach with unclear benefits for both developers and consumers. As a result, this issue becomes more and more topical in a context of REST-services orchestration based on the corresponding descriptions.

Purpose

The purpose of this work is to explore existing solutions in creating descriptions and composition of REST services as well as proposing own solution which would be more suituble for REST style and made it possible automatic service composition based on semantic descriptions understandeble for machine clients.

Solution

In order to solve set tasks, we propose the metamodel which implies two levels of interaction - semantic and activity. Service semantics was added by means of extending Schema.org vocabulary and usage new items in JSON. As a result, a path graph was created using generated descriptions, which helps us to find solution path in service composition. The proposed solution was tested on such services as Spotify

, Songkick and Uber. The results of the work are presented in a series of diagrams, tables, flowcharts with a detailed description, and explanations.

Object of research

The Semantic service orchestration.

Subject of research

Approaches to the creation of semantic descriptions and ways of orchestration of services on their basis.

Research methods

To solve the problem were used methods of analysis and synthesis, system analysis, comparison, logical generalization of results in this paper .

Scientific novelty

The scientific novelty of the work consists on creating a new model for solving the problems of describing and composing REST-services. With the further implementation of the proposed ideas and, analysis of the results.

The practical value of the results

The obtained results represent a new step towards the implementation of semantics in REST-services. They demonstrated the ability to use graph databases for these purposes. A broad field for further research and development was opened.

Research results approbation

Research results presented at the international scientific journal "Internauka", issue №8 2018.

Publications

Mahas V.V. Composition of semantic REST-services using Neo4J / V.V. Mahas. // International scientific magazine "Internet Science". - 2018 - №8. - P. 56-69.

Keywords

REST services, JSON, graph database, semantic orchestration, ontology, service description.

Зміст

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

JSON - JavaScript Object Notation

REST – Representational State Transfer

RDF – Resource Description Framework

HTTP - Hyper Text Transfer Protocol

URI – Universal Resource Identifier

NLTK - Natural Language Toolkit

ВСТУП

Існує два основні підходи до створення Web-сервісів. Один заснований на WSDL / SOAP і широко поширений в сценаріях B2B; інший REST-сервіси або Web API (Application Programmable Interfaces) широко поширені у Web. Web API - це популярний спосіб надання доступу до ресурсу, уникаючи складності стека технологій і стандартів, які вимагають сервіси створені на основі SOAP. REST сервіси, з іншого боку, вимагають додаткових обмежень, таких як узгодження контенту, відповідне використання мережевого протоколу і гіпермедіа.

Representational State Transfer(REST) (Fielding, 2000) - це архітектурний стиль, який лежить в основі Інтернету. Основними переваги якого є підтримка відмінної масштабованості та еволюційна здатність. REST-сурвіс являє собою набір ідентифікованих ресурсів, маніпуляція якими здійснюється посердництвом відомого набору методів, таких як Hypertext Transfer Protocol (HTTP). Семантика цих методи зрозуміла архітектурним компонентам (клієнтам, серверів, проксі, і т.д.).

Однак, починають виникати деякі обмеження, коли користувачем сервісу виступає машинний клієнт, а не веб-клієнт, керований людиною. Останній просто відкриває ресурси і залишає людину-користувача яка, як правило, розуміє що лежить в основі семантику на рівні бізнесу. Оскільки така семантика представлений на природній мові, машинний клієнт не може зрозуміти семантику такого типу. Як наслідок, машинний клієнт не може визначити необхідні дії для досягнення певної бізнес-цілі. Наприклад, клієнт повинен розуміти, що, переходячи за певним посиланням або подаючи певну форму, платіж виконано. Однак, обравши інше посилання, можна просто додати інший товар корзину для подальшої купівлі. Обидві дії розрізняються на рівні бізнес-семантики, але у на рівні веб-додатку - це просто методи HTTP POST на різних ресурсах.

Взаємодія між сервісами є вельми бажана, оскільки вона дозволяє повторне використання та компонування, забезпечує цілу низку дивідендів для бізнесу за відсутності потреби створювати сервіси з нуля. Такі привабливі перспективи мотивували постачальників сервісів публікувати семантику послуг на рівні бізнесу. В даний час популярний підхід Web Application Programming Interface (Richardson, 2013), який являють собою набір ресурсів і методів, які зазвичай документуються через редагування HTML сторінки. Ці документи

більш-менш докладно описують, які методи можуть бути виконані на ресурсі, необхідні параметри, обмеження на їх значення і очікувані результати на рівні бізнесу. Оскільки документація є adhoc, машинні клієнти повинні інкапсулювати всю логіку веб-API, необхідну для взаємодії з сервісом. Крім того, зміни в веб-інтерфейсі порушують роботу клієнта або, що ще гірше, призводять до виникнення невідповіностей. Відсутність зручного для зчитування машинами опису сервісів накладає значні обмеження на їх компонування. Більше того, складніші сценарії, як от динамічне компонування сервісів в рантаймі породжує суттєві складності .

Останнім часом, все більше пропозицій стосовно опису REST-сервісів пропонуються академічними колами, а недавно, деякі альтернативи були також висунуті в промисловості (наприклад, RAML , Swagger). Не зважаючи на те, що вони являються кроком вперед в напрямку розвитку стандартизованого опису REST-сервісів, поточні пропозицій операційно-орієнтовані, що ускладнює підтриму гіпермедіа і, як наслідок, обмежує автоматичне виявлення сервісів та їх еволюційність. Вони не підтримують,доступну для опрацювання машинним клієнтами, семантику сервісу на бізнес-рівні, що ускладнює їх подальше компонування.

В даній роботі була розроблена метамодель нового підходу, реалізована відповідна архітектуру, здатна аналізувати опису сервісів та генерувати на їх основі граф. Даний граф, у подальшому, використовується для створення запитів, що дозволяють автоматизувати процес компонування сервісів. Композиція REST-сервісів - це робочий процес(workflow) або шлях виконання методів, які виконуються на ресурсах і поєднаних певною схемою управління. Ми впровадили тестовий сценарій, використавши для цього три досить відомі веб-сервіси (Spotify , Songkick і Uber).

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Базові поняття Semantic Web

Семантична павутина (Semantic Web) - частина глобальної концепції розвитку мережі Інтернет, метою якої є реалізація можливості машинної обробки інформації, доступної у Всесвітній павутині. Основний акцент концепції робиться на роботі з метаданими, що однозначно характеризують властивості і зміст ресурсів Всесвітньої павутини, замість використовуваного в даний час текстового аналізу документів. Термін вперше введений сером Тімом Бернерс-Лі в травні 2001 року в журналі "Scientific American", і називається "наступним кроком в розвитку Всесвітньої павутини". У семантичній павутині передбачається повсюдне використання, по-перше, універсальних ідентифікаторів ресурсів (URI), а по-друге - онтологій і мов опису метаданих. Ця концепція була прийнята і просувається Консорціумом W3. Для її впровадження передбачається створення мережі документів, що містять метадані про ресурси Всесвітньої павутини і існуючої паралельно з ними. Тоді як самі ресурси призначені для сприйняття людиною, метадані використовуються машинами (пошуковими роботами та іншими інтелектуальними агентами) для проведення однозначних логічних висновків про властивості цих ресурсів.

Semantic Web - це еволюція World Wide Web, інформація в якій машинно-оброблювана (а не тільки орієнтована на обробку людиною), таким чином, дозволяючи браузерам або іншим програмним агентам здійснювати пошук, розподіляти і комбінувати інформацію набагато простіше. Semantic Web передбачає об'єднання цих різних видів інформації в єдину структуру, де кожному елементу "людської" інформації буде відповідати машинний код - спеціальний смисловий тег. Semantic Web в математичній формі являє собою

різновид графа - набору вершин, з'єднаних дугами. У Semantic Web роль вершин виконують поняття бази знань, а дуги (причому спрямовані) задають відносини між ними. Таким чином, семантична мережа відображає семантику предметної області у вигляді понять і відносин. Ідея полягає в тому, щоб глобальної семантичної мережею було підмножина систем, які замкнуті на специфічних шляхах досягнення достатнього зручності для машин. Таким чином, Семантична Мережа сама собою не буде задавати виводить машину. Вона буде задавати валідність операції і вимагати зв'язків між ними.

Розглянемо стан сучасної глобальної мережі і принципи роботи сучасних пошукових систем.

Подання інформації в мережі:

- теги в HTML не несуть семантичного навантаження;
- відсоток корисної інформації менше відсотка розмітки;
- розмітка, в тому числі, через велику складність і вкладеність (наприклад, проблема табличної верстки) містить багато помилок.

Інформація призначена тільки для перегляду людиною, з чого випливають такі принципи роботи пошукових систем сьогодні:

- вагові коефіцієнти на основі розташування слів;
- важливість слова в залежності від тега;
- релевантність (тобто скільки разів зустрічається дане слово в даному документі по відношенню іншим);
- аналіз "ваги" посилань в залежності від кількості посилань вказують на дану сторінку.

Через це виникають такі проблеми:

- машини не розуміють і, отже, не аналізують зміст інформації
- пошук незручний і складний, часто результати незадовільні і не релевантні;

- оптимізатори (SEO - Search Engine Optimizations) "грають" на недосконаліть алгоритмів пошукових систем, навмисне порушуючи правильність розмітки для короткочасного ефекту високих позицій в пошукових запитах.

Але є і ряд позитивних тенденцій, які дозволяють практично наблизитися до Semantic Web:

- з'явилася можливість ефективно відокремлювати розмітку від оформлення шляхом застосування CSS;
- на зміну HTML прийшла на заміну різновид XML мови опису документів - XHTML.
- Для того щоб вирішити всі перераховані вище проблеми Консорціумом W3 рекомендував рішення - застосування Semantic Web.

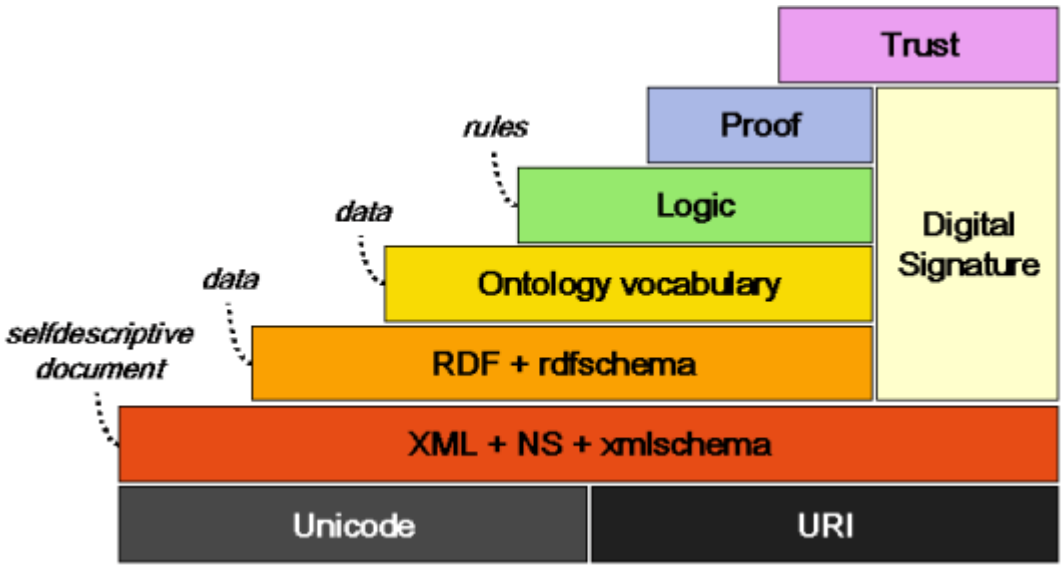

Рисунок 1.1 – Структура семантичної павутини[3]

Машинна обробка можлива в семантичній павутині завдяки двом її найважливіших характеристик:

1. Повсюдне використання універсальних ідентифікаторів ресурсів (URI).
2. Повсюдне використання онтологій і мов опису метаданих.

Тут варто зазначити, що пошукові агенти отримають можливість взаємодіяти не тільки з інформацією, що зберігається в мережі і доступною їй для обробки, але ще і між собою. Це дає можливість, як перевіряти результат, отриманий одним агентом, так і знаходити більш якісне рішення, а також уточнювати отримані результати.

Технічну частину Semantic Web становить сімейство стандартів на мови опису, що включає XML, XML Schema, RDF, RDF Schema, OWL, а також деякі інші. Маючи в своєму розпорядженні їх в порядку підвищення рівня абстракції, що реалізується тією чи іншою мовою, отримуємо.

XML надає синтаксис для визначення структури документа, що підлягає машинній обробці. Синтаксис XML не несе семантичного навантаження.

XML Schema визначає обмеження на структуру XML-документа, для того, щоб забезпечити передбачуваність обробки. Стандартний синтаксичний аналізатор мови XML в змозі перевірити довільний XML-документ на відповідність його структури, так званою схемою документа, описаної в XML Schema.

RDF являє собою простий спосіб опису екземплярність даних в форматі суб'єкт-відношення предикат-об'єкт, в якому в якості будь-якого елементу цієї трійки використовуються тільки ідентифікатори ресурсів. Існує стандартизоване відображення цих трійок на XML-документи зумовленої структури (тобто консорціумом W3 визначена схема XML -документів, що містять RDF-опис), а також на інші формати представлення.

RDF Schema (RDF-S) описує набір атрибутів (тут їх точніше назвати відносинами), таких, як rdfs: Class, для визначення нових типів RDF -даних. Мовою підтримується також ставлення успадкування типів rdfs: subClassOf. Таким чином, RDF Schema описує властивості, класи і ієрархії ресурсів RDF.

OWL (Web Ontology Language) розширює можливості по опису нових типів (зокрема, додаванням перерахувань), а також дозволяє описувати нові

типи даних RDF Schema в термінах вже існуючих (наприклад, визначати тип, який є перетином або об'єднанням двох існуючих). OWL використовується для точного уявлення значень термінів у словниках і опису взаємозв'язків між цими термінами. Це уявлення термінів і їх взаємозв'язків називається онтологією. OWL має більше механізмів для вираження значень, ніж XML, RDF і RDF-S, і він перевершує ці мови по можливості представляти контент, який можуть інтерпретувати машини.

Онтології. У філософії онтологією називають теорію про природу буття і видах сутностей. Онтологічний рівень формалізує накопичені знання, визначаючи і об'єднуючи термінологію різних предметних областей.

Онтології отримали досить широке поширення в задачах представлення знань та інженерії знань, семантичної інтеграції інформаційних ресурсів, інформаційного пошуку і т.д. У науці про "штучний інтелект" онтологія - це "специфікація концептуалізації предметної області", або спрощено, документ або файл, що формально задає відносини між термінами. Це свого роду словник понять предметної області і сукупність явно виражених припущень щодо змісту цих понять.

Найчастіше онтологія представляється як ієрархія понять, пов'язаних відносинами деяких певних видів. Такі визначення онтологій використовуються в різних класифікаціях. Розвинені онтології формалізуються засобами мов логіки і допускають можливості логічного висновку.

У найпростішому випадку онтології можна використовувати для підвищення точності пошуку в Internet - пошукова система буде видавати тільки такі сайти, де згадується в точності шукане поняття, а не довільні сторінки, в тексті яких зустрілося задане ключове слово.

Загальновідомо, що в різних предметних областях одні й ті ж поняття можуть бути представлені різними термінами. Механізм онтологій в цих випадках дозволяє формувати осмислені ієрархічні взаємозв'язки між об'єктами, узагальнювати і спільно використовувати глобальні відомості, тобто

реалізувати нечіткий пошук, здатний знаходити навіть такі необхідні користувачу ресурси, в яких не буде жодного слова з вихідного запиту.

Передбачається, що "інтелектуальні" програми будуть використовувати онтології, щоб одержувану в результаті пошуку інформацію з пов'язаною з нею структурою знань і правилами виведення. Механізми пошуку можуть застосовувати онтології і для вибірки сторінок з синтаксично різними, але семантично однаковими словами. Онтології також можуть використовуватися для організації обміну даними та інтеграції програм.

Така інтелектуальна програма, що інтерпретує онтології, зможе вивести, наприклад, що якщо Університет Корнелла знаходиться в м. Ітака, який розташований в штаті Нью-Йорк, який, у свою чергу, є частина США, то адресу цього університету слід писати в американському форматі.

Розробка мови опису структурованих онтологій OWL стало останнім часом одним з найбільш важливих ланок робіт з Семантичний Web, що проводяться консорціумом W3C. В кінці 2001 року для цієї мети в складі W3C була заснована спеціальна робоча група - Web Ontology Working Group. 10 лютого 2004 WWW- Консорціум присвоїв мові OWL статус рекомендованої до реалізації технології. В рамках OWL онтологія - це сукупність тверджень, які задають відносини між поняттями і визначають логічні правила для міркувань про них. Комп'ютери можуть "розуміти" сенс семантичних даних на Web-сторінках, слідуючи за гіперпосиланнями, ведучим на онтологічні ресурси. Онтологія може включати описи класів, властивостей і їх приклади (індивіди).

Формальна семантика OWL описує, як отримати логічні висновки на основі онтологій, тобто отримати факти, які не представлені буквально, а випливають з семантики онтології. Ці висновки можуть базуватися на аналізі одного документа або безлічі документів, розподілених в Мережі. Останнє забезпечується можливістю онтологій бути пов'язаними, включаючи прямий імпорт інформації з інших онтологій.

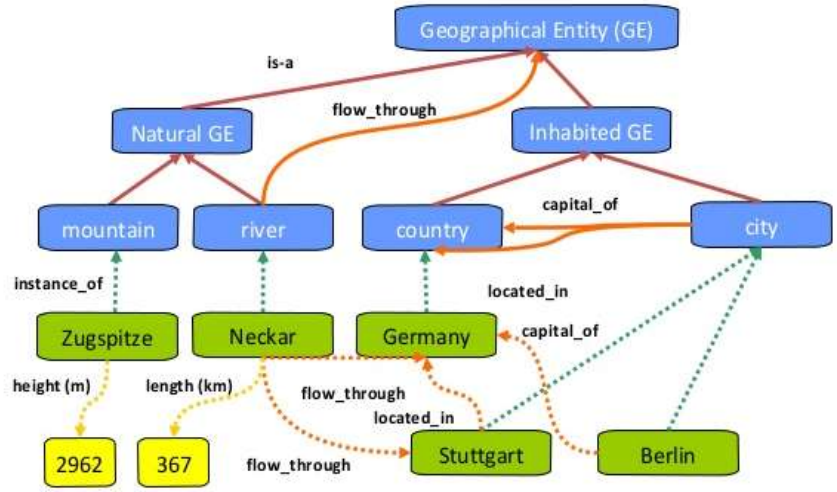

Рисунок 1.2 – Приклад онтології[6]

Щоб написати онтологію, яка може однозначно інтерпретуватися і використовуватися програмними агентами, задіюються синтаксис та формальна семантика OWL. На практиці створення онтологій починається з ієрархії класів понять, що складають предметну область. Фундаментальним таксономічним конструктором для класів є rdfs: subClassOf. Він пов'язує більш приватний клас з більш загальним класом. Якщо X - підклас Y, то кожен представник X - також представник Y. Ставлення rdfs: subClassOf є транзитивним. Якщо X - підклас Y, і Y - підклас Z, то X - підклас Z. Нижче наведені описи класів і підкласів, запозичені з рекомендацій W3C і пов'язані з предметною областю - виноробством.

<Owl: Class rdf: ID = "Напій">

<Rdfs: subClassOf rdf: resource = "# ПродуктХарчування" />

...

</ Owl: Class>

У даному прикладі Напій визначено як підклас класу ПродуктХарчування. Нижче дано спрощене визначення для класу Вино, яке є Напоєм:

<Owl: Class rdf: ID = "Вино">

<Rdfs: subClassOf rdf: resource = "Напій" />

<Rdfs: label xml: lang = "en"> wine </ rdfs: label>

<Rdfs: label xml: lang = "ru"> вино </ rdfs: label>

<Rdfs: label xml: lang = "fr"> vin </ rdfs: label>

...

</ Owl: Class>

Конкретний об'єкт в OWL Прийнято називати індивідом. У прикладі ВиноградКаберне Совіньон - це індивід, бо позначає один конкретний сорт винограду:

<Owl: Class rdf: ID = "ВинноВиноград">

<Rdfs: subClassOf rdf: resource = "& food; Виноград" />

</ Owl: Class>

<ВинноВиноград rdf: ID = "ВиноградКаберне Совіньон" />

В OWL існує безліч способів обмеження відносин. Наприклад, можна визначити домен і діапазон, - так властивість може бути визначена як підвластивість (спеціалізація) вже існуючої властивості.

<Owl: ObjectProperty rdf: ID = "зробленоЗВинограду">

<Rdfs: domain rdf: resource = "# Вино" />

<Rdfs: range rdf: resource = "# Виноград" />

</ Owl: ObjectProperty>

У даному прикладі властивість зробленоЗВинограду має домен Вино і діапазон Виноград. Таким чином, це пов'язує представників класу Вино з

представниками класу Виноград[2]. Множинні домени означають, що доменом властивості служить перетин зазначених класів.

Бази знань. База знань є одним з ключових компонентів інтелектуальних систем різного призначення. Розробка цього компонента є трудомістким і тривалим процесом. При розробці баз знань важливо забезпечити не тільки можливість зберігання знань і навігації по ній, а й можливість роботи над створенням і зміною бази знань розподіленим колективом розробників.

В основі розробки баз знань лежить чіткий поділ процесу проектування формального опису семантичної моделі розробляється бази знань від процесу реалізації (інтерпретації) цієї моделі на тій чи іншій платформі. Даний факт дозволяє забезпечити крос-платформну розробку інтелектуальних систем. Всю семантичну мережу (максимальну семантичну мережу), збережену в семантичної пам'яті абстрактної логіко-семантичної моделі інтелектуальної системи, будемо називати абстрактною семантичною моделлю бази знань цієї інтелектуальної системи. Семантична модель бази знань інтелектуальної системи є формальним трактування семантичного простору, яке відоме інтелектуальній системі в поточний момент часу. Ефективність інтелектуальної системи в першу чергу визначається обсягом і якістю формалізованих експертних знань, як декларативних (теоретичних), так і процедурних (практичних навичок) які містяться в ній .

База знань повинна містити в собі всю інформацію, необхідну агентам, які працюють над семантичною пам'яттю, для організації колективної діяльності щодо вирішення завдань, з якими повинна справлятися інтелектуальна система. Для розширення різноманітності видів знань, що зберігаються в базі, необхідним етапом в розробці семантичної моделі бази знань є її структуризація. Поняття бази знань тісно пов'язане з поняттям предметної області[9]. Співвідношення між базою знань і описуваної нею предметною областю задає семантику бази знань інтелектуальної системи. Розгляд структури бази знань у взаємозв'язку з предметною областю дозволяє

розглядати досліджувані об'єкти на різних рівнях деталізації. Деталізацію розгляду досліджуваних об'єктів можна здійснювати як в рамках вихідної предметної області, так і в системі самостійних, але пов'язаних між собою предметних областей.

При переході від предметної області до її моделі, представленої у вигляді семантичної мережі, виконуються наступні умови:

- кожному елементу предметної області взаємно однозначно відповідає позначає його елемент семантичної мережі;
- кожному сигнатурному елементу предметної області взаємно однозначно відповідає або позначає його ключовий вузол семантичної мережі, або позначає елемент алфавіту семантичної мережі.

Першим і найважливішим етапом проектування семантичної моделі бази знань є уточнення структури описуваної предметної області або декількох взаємопов'язаних предметних областей. Під уточненням структури предметної області розуміється явне виділення класу досліджуваних об'єктів, класу вторинних об'єктів, побудованих на основі досліджуваних, класу допоміжних об'єктів, через зв'язки з якими описуються деякі характеристики досліджуваних об'єктів, відносини, зв'язки яких пов'язують тільки досліджувані об'єкти між собою, а також відносини, зв'язки яких пов'язують досліджувані об'єкти з допоміжними.

Розгляд бази знань з позиції її співвідношення з предметною областю дозволяє розглядати досліджувані об'єкти на різних рівнях деталізації:

- класифікація класу досліджуваних об'єктів за різними ознаками;
- класифікація самих досліджуваних об'єктів, тобто розгляд структур взаємопов'язаних частин цих об'єктів;
- розгляд зв'язків досліджуваних об'єктів з допоміжними об'єктами, що не входять в клас досліджуваних об'єктів.

Залежно від досліджуваних об'єктів можна говорити про досить широку топологію предметних областей. Можна виділити наступні класи предметних областей:

- предметна область, що описує теоретико-множинні характеристики і зв'язку заданого сімейства об'єктів. Такі предметні області, зокрема, можуть бути онтологіями інших предметних областей;
- термінологічна онтологія - це клас предметних областей, для кожної з яких об'єктами дослідження є терміни (словосполучення), які відповідають різним елементам описуваної предметної області;
- предметна область, що є логічним описом деякої предметної області - це клас предметних областей, для кожної з яких об'єктами дослідження є всілякі логічні формули, що описують причинно-наслідкові закономірності, що інтерпретуються на деякій описуваної предметної області;
- логічна система понять, описуваних в заданій формальної теорії. Ця предметна метаобласть виділяє клас понять, не обумовлених в заданій формальної теорії, і пов'язує кожне визначається поняття з тими поняттями, на основі яких воно визначається;
- логічна система тверджень заданої формальної теорії. Ця предметна мтеаобласть виділяє клас аксіом для заданої формальної теорії, кожної теоремі ставить у відповідність одне з її доказів (основний доказ) і пов'язує кожну теорему з усіма тими твердженнями і визначеннями, які використовуються в основному доказі цієї теореми;
- предметна область питань та інформаційних задач - це клас предметних областей, для кожної з яких об'єктами дослідження є питання, інформаційні завдання, узагальнені питання і узагальнені інформаційні завдання, поставлені по відношенню до деякої описуваної предметної

області, а також відповідні їм способи вирішення інформаційних завдань (то є різні програми).

Семантична структура бази знань інтелектуальної системи трактується в рамках технології проектування баз знань інтелектуальних систем як ієрархічна система взаємопов'язаних між собою предметних областей, які представляються в базі знань[8].

На безлічі предметних областей можуть бути задані наступні відносини: включення, об'єднання, перетин, декомпозиція, гомоморфізм, ізоморфізм, теоретико-множинна онтологія, логічне опис, логічна онтологія. Виходячи з цього, ми можемо розглядати якусь метаобласть, об'єктами дослідження якої є всілякі предметні області. Таким чином, семантична структура бази знань являє собою ієрархічну систему описуваних нею предметних областей, що надбудовуються над заданою основний предметною областю.

Побудова семантичної структури бази знань вимагає не тільки явного уявлення специфікації кожної описуваної предметної області у вигляді формального тексту, а й явного опису всіляких зв'язків між цими предметними областями.

З метою скорочення часу процесу проектування семантичних моделей баз знань інтелектуальних систем необхідно створити бібліотеку семантично сумісних компонентів баз знань. На основі цієї бібліотеки розроблена методика компонентного проектування баз знань.

До основних типів компонентів баз знань, що зберігаються в бібліотеці відносяться:

1. онтології різних предметних областей, які можуть бути найрізноманітнішими за змістом, проте повинні бути семантично сумісними;
2. базові фрагменти теорій, які відповідають різним рівням знання користувача, починаючи від базового шкільного до професійного;
3. семантичні околиці різних об'єктів;

4. специфікації формальних мов опису різних предметних областей.

Для забезпечення семантичної сумісності таких компонентів баз знань, які є уніфікованими семантичними моделями, необхідно:

- узгодити семантику всіх використовуваних ключових вузлів;
- узгодити глобальні ідентифікатори ключових вузлів, використовуваних в різних компонентах. Після цього інтеграція всіх компонентів, що входять до складу бібліотеки, і в будь-яких комбінаціях здійснюється автоматично, без втручання розробника.

Для включення компонента в бібліотеку необхідно його уточняти за наступними критеріями:

- предметна область, опис якої міститься в компоненті;
- клас (тип) компонента бази знань;
- склад бази знань;
- кількісні характеристики ключових вузлів бази знань;
- інформація про розробників бази знань;
- дата створення бази знань;
- інформація про верифікації бази знань;
- версія компонента бази знань;
- умови поширення компонента бази знань;

1.2 Особливості RESTful web-сервісів

REST визначає ряд архітектурних принципів проектування Web-сервісів, орієнтованих на системні ресурси, включаючи способи обробки і передачі станів ресурсів по HTTP різноманітними клієнтськими додатками, написаними на різних мовах програмування. За останні кілька років REST стала переважаючою моделлю проектування Web-сервісів. Фактично REST зробила

настільки великий вплив на Web, що практично витіснила дизайн інтерфейсу, заснований на SOAP і WSDL, завдяки більш простому стилю проектування.

Технологія REST не притягнула велику увагу в 2000 році, коли Рой Філдінг вперше представив її в Каліфорнійському університеті в Ірвайні в своїй дисертації "Архітектурні стилі і дизайн мережевих архітектур програмного забезпечення", де аналізувався набір принципів архітектури програмного забезпечення, що використовує Web в як платформа розподілених обчислень. Однак сьогодні, через багато років, виникли і продовжують розвиватися численні інфраструктури для REST[5].

Ми припускаємо, що в чистому вигляді (в якому вона привертає таку пильну увагу) конкретна реалізація Web-сервісів REST слідує чотирьом базовим принципам проектування:

- Явне використання HTTP-методів.
- Незбереження стану.
- Надання URI, аналогічних структурі каталогів.
- Передача даних в XML, JSON або в обох форматах.

Далі розглянемо ці чотири принципи і приведемо технічне обгрунтування їх важливості для розробників Web-сервісів REST.

Явне використання HTTP-методів. Однією з ключових характеристик REST-сервісу є явне використання HTTP-методів згідно з протоколом, визначеним в RFC 2616. Наприклад, HTTP GET визначається як метод генерування даних, використовуваний клієнтським додатком для вилучення ресурсу, отримання даних з Web-сервера або виконання запиту в надії на те, що Web-сервер знайде і поверне набір відповідних ресурсів.

REST пропонує розробникам використовувати HTTP-методи явно відповідно до визначення протоколу. Цей основний принцип проектування REST встановлює однозначну відповідність між операціями create, read, update і delete (CRUD) і HTTP-методами. Згідно з цим відповідності:

Для створення ресурсу на сервері використовується POST.

Для вилучення ресурсу використовується GET.

Для зміни стану ресурсу або його поновлення використовується PUT.

Для видалення ресурсу використовується DELETE.

Незбереження стану. Для задоволення постійно зростаючих вимог до продуктивності REST-сервіси повинні бути масштабованими. Для формування топології сервісів, що дозволяє при необхідності перенаправляти запити з одного сервера на інший з метою зменшення загального часу реакції на виклик Web-сервісу, зазвичай застосовують кластери серверів з можливістю розподілу навантаження і аварійного перемикання на резерв, проксі-сервери і шлюзи. Використання проміжних серверів для поліпшення масштабованості вимагає, щоб клієнти REST-сервісів відправляли повні самодостатні запити, що містять всі необхідні для їх виконання дані, щоб компоненти на проміжних серверах могли перенаправляти, маршрутизувати і розподіляти навантаження без локального збереження стану між запитами.

При обробці повного самодостатнього запиту серверу не потрібно витягувати стан або контекст програми. Додаток REST-сервісу включає в HTTP-заголовки і в тіло запиту всі параметри, контекст і дані, необхідні серверному компоненту для генерування відповіді. У цьому сенсі незбереження стану (statelessness) покращує продуктивність Web-сервісу і спрощує дизайн і реалізацію серверних компонентів, оскільки відсутність стану на сервері усуває необхідність синхронізації сеансових даних із зовнішнім додатком.

Відображення URI, аналогічних структурі каталогів. З точки зору звернення до ресурсів з клієнтського додатку відображені URI визначають, наскільки інтуїтивним буде REST-сервіс і чи буде він використовуватися так, як припускав розробник. Третя характеристика RESTful-сервісу повністю присвячена URI[11].

URI-адреси Web-сервісу REST повинні бути інтуїтивно зрозумілими. Розглядайте URI як якийсь інтерфейс з властивістю самодокументування, що

майже не вимагає пояснень або звернення до розробника для його розуміння і для отримання відповідних ресурсів. Тому структура URI повинна бути простою, передбачуваною і зрозумілою.

Один із способів досягти такого рівня зручності використання - побудова URI за аналогією зі структурою каталогів. Такого роду URI є ієрархічними, що походить із одного кореневого шляху, розгалуження якого відображають основні функції сервісу. Згідно з цим визначенням, URI - це не просто рядок з "/" як роздільником, а скоріше дерево з гілками, з'єднаними в вузлах. URI повинні бути статичними, щоб при змінах ресурсів або реалізації сервісу посилання залишалася тим же. Це дозволяє зберегти закладки. Також важливо, щоб взаємозв'язки між ресурсами, закодованими в URI, залишалися незалежними від способу вказівки місця розташування ресурсів в сховищі.

Передача даних в XML, JSON або в обох форматах. Подання ресурсу, як правило, відображає поточний стан ресурсу (і його атрибутів) на момент його запиту клієнтським додатком. Представлення ресурсів в цьому сенсі є просто знімками в конкретні моменти часу. Ці представлення повинні бути такими ж простими, як представлення записів в базі даних, що складається з відображення між іменами стовпців і XML-тегами, де значення елементів в XML містять значення рядків. Якщо система має модель даних, то згідно з цим визначенням представлення ресурсу є знімком стану атрибутів одного з об'єктів моделі даних системи. Це ті об'єкти, які буде обслуговувати REST-сервіс.

Останній набір обмежень, тісно пов'язаний з дизайном RESTful-сервісів, відноситься до формату даних, якими обмінюються додаток і сервіс при роботі в режимі запит / відповідь або в тілі HTTP-запиту. Тут особливо важливі простота, читабельність і зв'язаність.

Об'єкти моделі даних зазвичай якось пов'язані, і ці відносини між об'єктами (ресурсами) моделі даних повинні відображатися в способі їх подання для передачі клієнтського додатку. Щоб надати клієнтським додаткам можливість запитувати конкретний найбільш підходящий їм тип вмісту, проектують сервіс

так, щоб він використовував вбудований HTTP-заголовок Accept, значення якого є MIME-типом. Це дозволить використовувати сервіс клієнтським додаткам, що написані різними мовах і працюють на різних платформах і пристроях. Використання MIME-типів і HTTP-заголовка Accept є механізмом узгодження вмісту (content negotiation), що дозволяє клієнтським застосуванням вибирати відповідний для них формат даних і мінімізувати зв'язність даних між сервісом і додатками, що його використовують.

1.3 Семантичне оркестрування сервісів

Оркестрування - автоматичне розміщення, координація і управління складними комп'ютерними системами і службами.

Оркестрування описує те, як сервіси повинні взаємодіяти між собою, використовуючи для цього обмін повідомленнями, включаючи бізнес-логіку і послідовність дій. Оркестрування підпорядковане якомусь одному з учасників бізнес-процесу.

Зв'язок між ресурсами в гетерогенній системі - дуже важлива і важка задача. У SOA він часто виступає як композиція сервісів, які більш-менш автоматизовані. Ми можемо розглянути різні способи взаємодії сервісів. Більш простим способом є ручний опис всіх взаємодій, які часто називають оркестровкою. Багато специфічних мови, такі як мови XML Process Definition Language (XPDL), Business Process Execution Lan-guage (BPEL) або Business Process Modeling Notation (BPMN), можуть виражати взаємодію між сервісами але є статичними і не можуть підтримувати динамічну зміну ресурсів, таких як заміна або оновлення сервіса. Хорошим рішенням є автоматизоване компонування сервісів без явного статичного опису. Але ніякі реальні ефективні методи не пропонують загального і корисного рішення. Для цього використовується спільний підхід - використання семантики для опису сервісів

і взаємодії між ними. Композиція може динамічно змінюватися, якщо необхідні їй концепції присутні в семантичному описі, часто представленому як онтологія. У хмарній архітектурі композиція також є невід'ємною частиною, а використання семантики - цікава задача, яка як і раніше є роботою в прогресі у багатьох дослідницьких групах.

Як видно на рисунку 1.3, перший рядок представляє статичне оркестрування BPEL. Деякі важливі маркери:

- process: кореневий елемент процесу,
- partnerLink: зв'язування сервісів з процесом,
- invoke: виклик веб-сервісу,
- variables: оголошення змінних для управління даними,
- assign: оновлення вмісту змінних.

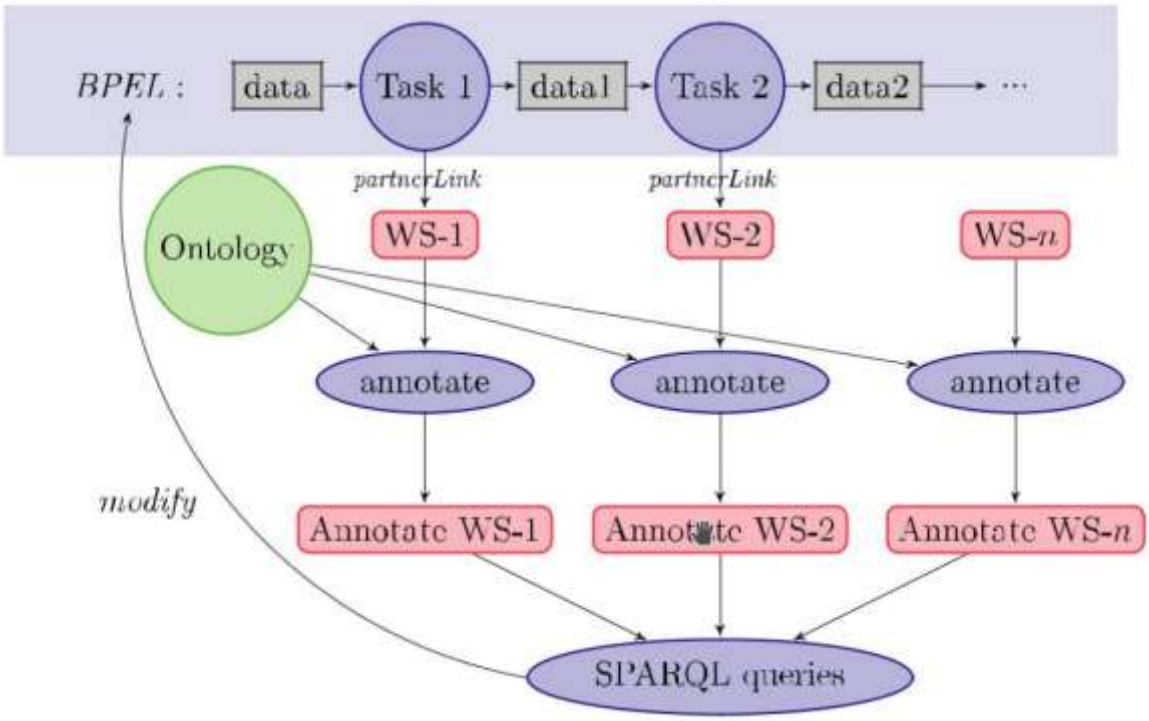

Рисунок 1.3 – Модель семантичного оркестрування[16]

1.4 Висновки

У даному розділі було розглянуто ключові поняття предметної області. Перш за все, розглянуто онтології та бази знать, які виступають наріжними каменями у створенні семантичних сервісів. Згодом було з’ясовано, що подання системних ресурсів через RESTful API - це гнучкий спосіб забезпечення різних додатків даними в стандартному форматі. Це допомагає виконати вимоги до інтеграції, що мають вирішальне значення для створення систем, що дозволяють легко комбінувати дані, а також для побудови на основі набору базових RESTful-сервісів більших конструкцій. У заключному підрозділі ми познайомилися з оркеструванням, та розглянули базову модель семантичного оркестрування.

2 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО СТВОРЕННЯ І КОМПОНУВАННЯ СЕМАНТИЧНИХ REST-СЕРВІСІВ

2.1 Додавання семантичного опису до REST-сервісів

Традиційно веб-сервіси описуються WSDL (Web Service Description Language), документом створеним на основі XML. WSDL описує інтерфейс сервісу (операції, параметри і URL) і умови його використання. Обмін повідомленнями між клієнтами і сервісами, повинен виконуватися відповідно до обмежень Simple Object Access Protocol (SOAP) схеми.

Таким чином SOAP-сервіси орієнтовані на операції, тоді як для REST-сервісів ключовим поняттям є ресурси. Ресурси ідентифікуються *ідентифікаторами ресурсів* (Uniform Resource Identifiers); пердставлення ресурсів являють собою набір байтів, що передає інформацію про стан ресурсу в певний час.

Ще одним елементом REST є обмеження *Uniform Interface*. Він визначає, що REST компоненти повинні підтримувати один і той же інтерфейс, керувати ресурсами через представлення (отримання або оновлення стану ресурсу з новим представленням), та що REST-компоненти (клієнти, сервери) взаємодіють завдяки повідомленням що містять свій опис (HTTP-повідомлення) і включають всю інформацію, необхідну для обробки такого повідомлення. Не менш важливо, що движок REST-додатку повинен слідувати *обмеженням гіпермедіа*, тобто представлення ресурсу має містити необхідні елементи управління (наприклад, кнопку відправки) і посилання, які повідомляють клієнту про доступний набір дій в поточному стані додатку.

У SOAP-сервісах одна кінцева точка інкапсулює довільну кількість призначених для користувача операцій чия симантика, попередні і пост-умови

визначаються кожним постачальником сервісу індивідуально. У REST сервіс являє собою набір ідентифікованих ресурсів, якими можна маніпулювати за допомогою чітко визначеного набору методів (методів HTTP), що полегшує еволюцію шляхом сервісів використанням веб-стандартів (наприклад, форматів даних, мережевих протоколів і т.д.) і обслуговування масштабованість, використовуючи архітектурні обмеження REST.

На відміну від традиційних SOAP-сервісів ресурси REST є явними, методи кінцеві та представлення чітко укладають в собі фрагменти бізнес-процесу. Обмеження для машинних клієнтів полягає в тому, що семантика таких елементів не визначена для машинного опрацювання на бізнес рівні.

Пропозиції стосовно зручного опису сервісу для машин, який би інкапсулював семантику сервісу були зроблені спочатку для SOAP-сервісів[4]. Прикладом такого опису може слугувати, ***OWL-S*** що була запропонована в 2004 році і грунтується на онтологічних моделях. Онтологія є формальною і явна специфікація загальної концептуалізації певного домену. Під нею, як правило, розуміють набір понять, їх властивостей, а також взаємини між ними. OWL-S пропонує набір онтологій, що описують домен сервісу (наприклад, банківська справа), механізми веб-сервісу (наприклад, операції відповідно до WSDL), протокол взаємодії (відповідно до стандартів SOAP) тощо.

Характерно, що онтологія OWL-S складається з трьох основних частин: сервісного профілю, моделі процесу і заземлення.

- Профіль сервісу використовується для опису того, що робить служба. Ця інформація призначена для зручності і включає ім'я та опис служби, обмеження на застосування і якість обслуговування, видавця і контактну інформацію.
- Модель процесу описує, як клієнт може взаємодіяти з сервісом. Це опис включає в себе набори входів, виходів, попередні умови і результати виконання служби.

- Заземлення вказує деталі, які клієнт повинен взаємодіяти з сервісом, якіпротоколи зв'язку, формати повідомлень, номери портів тощо.

З одного боку, OWL-S є досить виразна й багата концептуалізація сервісного домену, широко використовувана в семантичних дослідження. З іншого боку, OWL-S дуже складний і не підходить сервісній моделі REST (гіпермедіа та HTTP-методи не підтримуються).

Інший, більш легкий підхід, ***SAWSDL*** розроблений в 2007 році, є рекомендацією W3C для опису семантичного сервісу, що складається з мінімального набору елементів, які можуть використовуватися для анотування стандартного WSDL. Підхід полягає в тому, щоб включати посилання (URI) до концепцій описаних в окремій онтології. Відчутними недоліком стандарту є те, що він не визначає ні онтології домену, ні семантичної моделі сервісу, ні мови представлення онтологій.

У 2006 році було запропоновано мову для опису REST-сервісів ***Web Application Description Language (WADL)***. Цей підхід еквівалентний WSDL для SOAP-сервісів і передбачає анотування семантичними посиланнями слідуючи підходу SAWSDL[19]. Сервіси будуть взаємодіяти за допомогою запитів SPARQL, і автори пропонують зіставлення між методами HTTP і SPARQL. Однак семантика таких дій не розглядається на бізнес-рівні. Ця пропозиція також вимагає,від розробника написання всієї бізнес-логіки для взаємодії з сервісом на стороні клієнта. Подібно WSDL, WADL піддається критиці за надмірну складність і багатослівність при тому повторює WSDL-орієнтований підхід і ігнорує обмеження гіпермедіа REST.

Інші підходи, такі як ***hREST***, пропонують HTML-мікроформат для анотування HTML-сторінки, який зазвичай використовуються для опису веб-API. Основне завдання мікроформатів визначається, як створення стандартних шляхів структурування тегів HTML-сторінки для опису семантичної

інформації, яка дозволила б ефективно опрацьовувати сторінку як людині, так і машинному агенту. Основні характеристики можна виглядають так:

- Використання HTML-структури і словника для визначення семантики
- Вбудованість в HTML, XHTML, RSS, XML
- Концентрація на простоті, мінімалізмі та повторному використанні

Однак, семантика дій не розглядається, і підхід навряд чи впорається зі складністю існуючих веб-інтерфейсів (наприклад, додаткові параметри, додатковий тип носія
відповіді, метадані тощо.).

Значним кроком в перед, у цьому сенсі, слугує підіхд до опису сервісу ***ReLL*** запропонований у 2010 році, що повністю враховує принципи REST. В експерименті він використовувався пошуковиком, щоб переміщатися по ресурсам деяких REST-сервісів, демонструючи можливості використання опису, враховуючи обмеження REST. Семантичний опис ReLL був отриманий через додатковий шар, який був використаний для отримання семантичного набору даних, еквівалентного шуканим даними[17]. ReLL розглянув один тип дії, метод GET і припускав одну дію такого роду (тобто читання стану ресурсів), а інтерфейс сервісу також простий (набір параметрів), але він розглянув семантику представлення і управління гіпермедіа.

HAL - це мова опису JSON, яка фокусується на гіпермедіа з урахуванням тільки методу GET. ***Hydra*** йде далі шляхом розгляду ресурсів, операцій і гіперпосилань, представлених у вигляді шаблонних посилань. Ці шаблони є класом властивостей, який пов'язує певну операцію з шаблонами IRI приєднуючи до набору підтримуваних змінних (параметри URI). Значення параметрів визначаються також під час виконання. Hydra грунтується на концепції JON-LD який додає легку семантику до опису.

Основна ідея Hydra полягає в наданні словника, який дозволяє серверу рекламувати дійсні переходи стану для клієнта. Потім клієнт може

використовувати цю інформацію для створення HTTP-запитів, які змінюють стан сервера, щоб досягти певної мети. Оскільки вся інформація про дійсні переходах стану обмінюється машинним способом під час виконання, а не записана в клієнті, клієнти можуть бути відокремлені від сервера і легше адаптуватися до змін. Однак ця пропозиція відзначається певною складністю через модель RDF, на якій вона також заснована.

Пропозиція, яка набула великої популярності називається ***Swagger*** (в даний час прийнята Open API в якості основної специфікації). Swagger може бути представлений як JSON або YAML формат і дозволяє описувати ресурси, операції і відповіді. Він забезпечує підтримку для задання параметрів операцій (необов'язкових чи обов'язкових) і схем відповідей простим і інтуїтивним способом. Основним його недоліком є те, що Swagger робить не підтримує семантичні асоціації з його елементами і гіпермедіа.

Подібна ініціатива це ***RAML***, заснований на YAML, який забезпечує додаткову підтримку за допомогою надання широкого набору типів даних, а також специфікацію URI, параметри запиту специфікації і різні схеми безпеки. RAML більш функціональний, але як наслідок складніший і менш інтуїтивний, ніж Swagger.

2.1 Семантичне оркестрування REST-сервісів

Композиція сервісів зазвичай розглядається як комбінація сервісних операцій, що слідують певному порядку виконання. Якщо сервіс не залежить від інших, тобто здатен самостійно завершити своє виконання, то він вважається атомарним ,у іншому випадку - зкомпонованим.

Композиція сервісів може бути *статичною*, якщо композиційна модельвизначається під час розробки сервісу або *динамічною*, якщо вона визначається під час виконання. Композиційна модель являє собою представлення набору сервісів, які викликаються, а також дані і управління, що

визначають взаємодію. *Динамічний* підхід полегшує компонування, коли є багато доступних сервісів для використання, що дозволяє скоротити витрати на розробку і час. Він також може сприяти швидкій реакції на невдачу, зміні бізнес-пріоритетів чи персоналізацію. Статична чи динамічна композиція може бути визначена автоматично або вручну в залежності від того, чи буде
алгоритм повинен автоматично обирати сервіси і визначати граф управління/потоку даних.

Крім того, підхід до компонування сервісів поділяють на *оркестрування* та *хореографію*. Організація оркестровки являє собою єдиний централізований виконуваний бізнес-процес (оркестр), який координує взаємодію між різними сервісами. Організатор відповідає за виклик і об'єднання сервісів. До складу оркестрування входить управління транзакціями між окремими службами. Хореографія сервісів - це глобальний опис сервісів, який визначається обміном повідомленнями, правилами взаємодії і угодами між двома або більше кінцевими точками. У хореографії використовується децентралізований підхід.

У REST ресурси і колекції ресурсів є компонентами. Сервери впроваджують в представлення ресурсів необхідний набір можливих посилань і елементів управління
для переходу стану. Цей підхід гіпермедіа нагадує хореографію, в якій клієнти і сервери співпрацюють для фактичного виконання певних бізнес-задач. Природно, що така координація може бути реалізована, як єдиний сервіс таким чином, забезпечуючи виконання бізнес-процесу. Однак, цей підхід серйозно ставить під загрозу масштабованість сервісу.

Для реалізації композиції REST-сервісів була запропонована ціла низка стратегій. Наприклад, **Bite** - мінімалістична хореографічна мова, побудована для компонування сервісів. Bite забезпечує середовище розробки для серверних скриптів для всіх видів додатків, які взаємодіють з браузером, пошовими клієнтами, ресурсами REST, віддаленими функціями, доступними через URL, JSON-RPC тощо. Bite зменшу витрати на розробку, застосовуючи підхід,

орієнтований на сценарій, в якому розробники можуть вибирати, які додаткові можливості використовувати відповідно до ситуації. Прості потоки даних можуть бути створені з використанням конструкцій мови, що також дозволяє підтримувати потужні тривалі асинхронні процеси, включаючи паралельну обробку[13].

JOpera слідує аналогічному підходу, але з використанням візуального моделювання для управління і потоку даних. Середовище візуальної розробки JOpera пропонує користувачам засоби для швидкої побудова процесів з існуючих бібліотек сервісних компонентів з подальшим моніторинг їх виконання. Наприклад, користувач може шукати зовнішні сервіси, переглядаючи через реєстр UDDI і імпортувати свої інтерфейси в бібліотеку. Це робиться шляхом перетворення описів інтерфейсу WSDL в візуальну нотацію JVCL: кожна операція сервісу імпортується як окремий дія. Для наглядності, з графом потоку даних можна ознайомитися на рисунку 2.1.

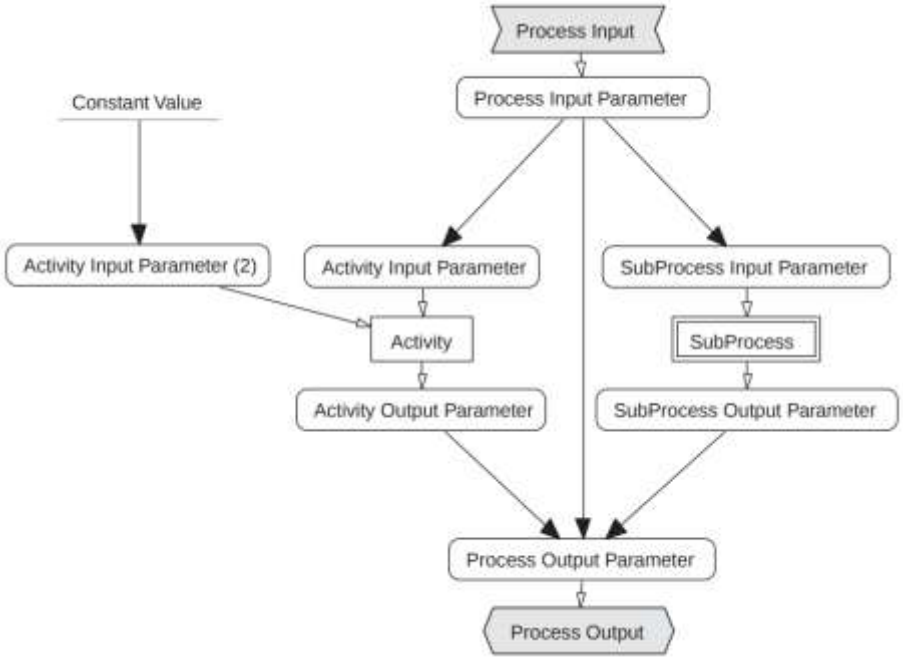

Рисунок 2.1 - Граф потоку даних в JOpera[7]

До слова, JOpera підтримує динамічне зв'язування сервісів, тоді як інші підходи вимагають статичної і ручної ідентифікації сервісних компонентів. Всі ці

підходи слідують централізованій стратегії оркестрування і розглядають, неявну семантику для ручного проектування композиції сервісів.

Семантичні підходи до машинного компонування REST-сервісів, грунтуються на на семантичних технологіях (наприклад, RDF, OWL, N3). Яскравим представником такої філософі є **SRSM**, який ідентифікує сервісну інформацію (об'єкти) і рівень транзакції для ресурсо-орієнтованої композиції сервісів. Його шари містять спеціалізовані ресурси а сервісна семантика, в свою чергу, через OWL онтологію. Для зв’язків між сервісами SRSM визначає формальний тип структури який називається Entity-Oriented Resource (EOR), що дозволяє компонувати малі ресурси в складені. Для сервісних операцій SRSM визначає інше тип структури, який називається Transition-Oriented Resource (TOR) для реалізації складних функції з ресурсами сервісів.

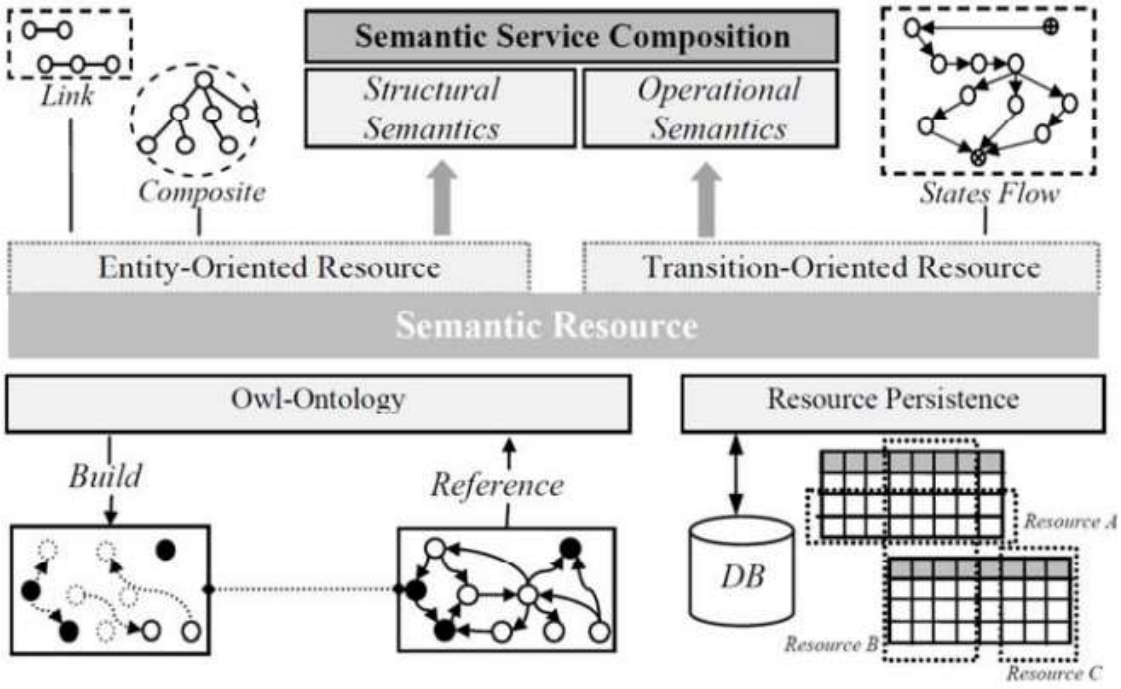

Рисунок 2.2 - Semantic Resource Service Model (SRSM)[6]

Для збереження ресурсу SRSM налаштовує інформацію про межі ресурсів, щоб чітко розрізняти різні ресурси на рівні фізичних даних. Для композиції сервісів рішення об'єднує структурну семантику і оперативну кожного з них, що чудово видно на рисунку 2.2. Автоматичне компонування

сервісів залежить від правил OWL-SWRL, що визначають попередні умови і ефекти, які повинні бути досягнуті за допомогою алгоритму пошуку.

RESTdesc пропонує використовувати мову Notation3(N3) і фреймворк N3 Logic для створення композиції REST-сервісів на RDF-ресурсах. Ідея полягає в тому, щоб визначити формули за допомогою N3, що складаються з: попередньої умови, класу HTTP-запиту і післяумови, які оцінюються з допомогою засобу N3 EYE, для створення ефективної композиції. З переваг Notation3, варто виділити додаткову підтримку графіків і кількісних оцінок, що дозволяє уникнути додаткових метарівневих конструкцій для семантики і функціональності. Як результат описи RESTdesc не вимагають явного звертання до сервісів, зазначення входів і виходів тощо[12]. Підсумовуючи, згадаємо ще кілька ключових аспектів RESTdesc описів:

- Вони не вводять ніякої нової термінології, а використовують повторно існуючі концепції, такі як N3 і URI [8]. В результаті, функціональність сервісу описується безпосередньо, тобто, він не вимагає додаткового опису для параметрів, налаштувань тощо.
- Немає необхідності вказувати типи, оскільки вони можуть бути виведені з онтологічних баз знань.
- Описи є самодостатніми і не вимагають додаткової семантики, крім наданої мовою N3.

Для повноти картини доречно розглянути приклад опису RESTdesc для окремого запиту (Рисунок 2.3).

```
@prefix book: <http://example.org/book#>.
{
  ?book book:reviewForm ?reviewForm.
}
=>
{
  _:request http:methodName "POST";
            http:requestURI ?reviewForm;
            http:body [tmpl:formData ("text=" ?text)];
            http:resp [tmpl:represents ?review].

  ?book book:review ?review.
  ?review book:reviewText ?text.
}.
```

Рисунок 2.3 - Опис RESTdesc [14]

Ще одним підходом, який вартий нашої уваги є запропонований Mahdi Bennara у 2014 році. Його суть полягає тому, що семантика REST-сервісу моделюються як пов'язаний ресурс. Цей ресурс містить як інформацію про сам сервіс так і переходи (підтримувані методи HTTP) з використанням JSON-LD. Композиція сервісів досягається за рахунок діалогового підходу, коли клієнт поступово перевіряє можливі дії, підтримувані обраним ресурсом. Цей підхід використовує легкі семантичні представлення, сприяє динамічному з'єднанню і еволюційності сервісів.

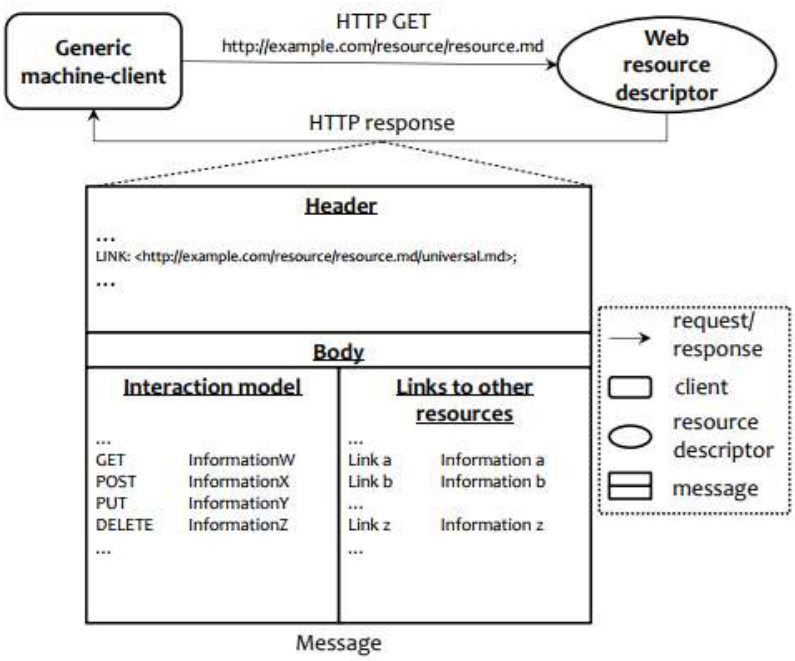

Рисунок 2.4 - Отримання опису ресурсів[20]

З іншого боку, клієнт не може заздалегідь знати які ресурси і транзакції, необхідні для досягнення певного стану. Тобто, якщо виявиться що такі методи, як POST, PUT або DELETE необхідні, вони не можуть бути скасовані. Тому клієнт повинен знати жорстко закодований необхідний шлях, хоча б, на семантичному рівні. Яскравий приклад роботи з описами в пропонованому підході наведено на рисунку 2.4.

2.3 Висновок

Підсумовуючи проведений аналіз досліджень в даній сфері, доречно виділити наступні моменти. Перш за все, для опису наших REST-сервісів точно не підійдуть стандартизовані підходи для додання семантики, як от *OWL-S*. Головним чином це пояснюється заточеністю під SOAP-сервіси та відсутністю підтримки гіпермедіа. Рішення ж такі як RAML, Swagger чи Hydra також не можуть повною мірою вирішити наше питання через окремі недоліки, серед яких RDF-орієнтованість, надмірна складність та відсутність підтримки

семантичних асоціацій. Розглянувши підходи до компонування REST-сервісів, оптимального варіанту теж знайти не вдалося. Найбільш досконалими можна назвати RESTdesc та SRSM, тож доречно буде використати окремі ідеї реалізовані в них для створення власного рішення.

3 ЗАПРОПОНОВАНИЙ ПІДХІД ДО ОПИСУ ТА ОРКЕСТРУВАННЯ СЕРВІСІВ

3.1 Опис сервісів

3.1.1 Метамодель

Наш підхід розділяє елементи REST-сервісів на два шари: семантики і активності (Рисунок 3.1). Семантичний рівень відображає зміст і призначення ресурсів, параметрів і дій. Шар активності містить елементи, які є реалізаціями семантичного шару.

Для семантичного шару **Resources, Parameters** і **Actions** є концепціями в бізнесі домені, який може бути семантично пов'язаний. Ці поняття є семантичним аспектом елементів в шарі активності, але вони не прив'язані до будь-якого формалізму представлення знань. Семантичний рівень пов'язаний з рівнем активності через реалізацію опису слідуючи підходу SAWSDL, без перевантаження опису. Семантичний шар використовується для поєднання різних сервісів, грунтуючись на їх призначенні та властивостях. Зверніть увагу, що **Actions** пов'язані як мінімум з одиним **Resource** і навпаки.

Рівень активності прив'язаний до REST-сервісів. Елементи **Resource** ідентифікуються їх URI і пов'язані з щонайменше одним елементом **Method** (наприклад, GET, PUT, POST, DELETE для HTTP-протоколу). Щоб виконати ці операції, елементу **Operation** можуть знадобитися елементи **Parameters**. Вхідні параметри можуть бути надані різними шляхами: як частина самого URI (змінні URI), заголовки, запит або його тіло. Параметри можуть бути повторно використані багатьма операціями. Після виконання операції, веб-сервіс повертає елемент **Response**, який повертає відповідь на запит. Ці відповіді також включають стан ресурсів у формі елемента **Representation.**

Representation складається з очікуваної інформації що повертається сервісом і може включати вихідні параметри і інші методи (гіпермедіа).

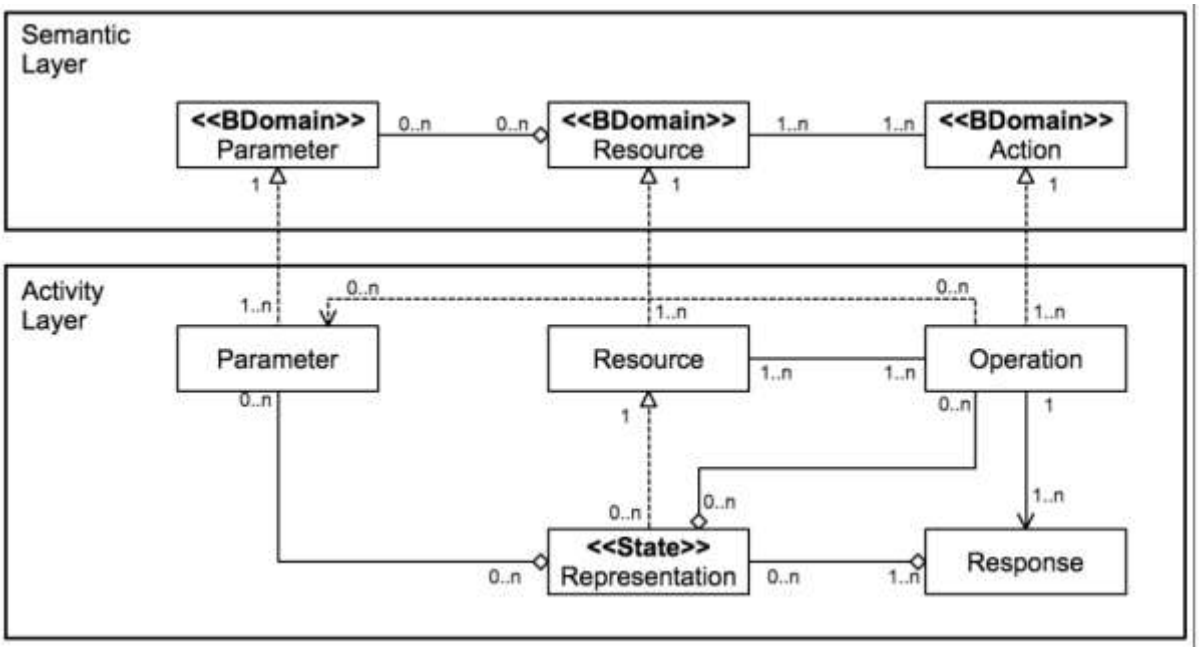

Рисунок 3.1 – Метамодель пропонованого рішення

3.1.2 Словник опису

Як описано вище, ресурси, параметри і дії пов'язані з концепціями в бізнес-домені, описаний в окремому документі, словнику. Словник пов'язує значення *reference* з URI, що описують поняття недвозначно. У нашому підході, ми розглянули і поширили концепції, визначені специфікацією Schema.org. Schema.org просувається Microsoft, Google та Yahoo і може використовуватися як HTML-розмітка, яка розширює фрагменти результатів пошуку. Специфікація передбачає безліч сутностей і механізмів розширення. У цій роботі ми використовуємо механізм, що дозволяє розширити концепцію, додавши властивості за шаблоном «BaseConcept / newProperty». Тобто, додаємо назву властивості, в нижньому регістрі, після назви концепції за символом «/». Аналогічним чином, клас можна визначити назвою спеціалізованої концепції, у

починаючи з верхнього регістру: «BaseConcept / SpecializedConcept». Словник був розроблений як документ JSON (рисунок 3.2).

Рисунок 3.2 - Фрагмент словника розробленого на основі Schema.org

Необхідними ключами, у даному випадку, є *name, version, baseUri* і *prefixes.* Префікси - це абревіатури концептуальних об'єктів таких як ресурси, параметри і дії. Вони повинні починатися з символу «@» та бути пов’язаними з явними URI через відповідний ключ. Кожна властивість об'єкта може бути визначено через ключ параметра, значення якого є фрагментом URI, який повинен бути доданий до базової концепції, як визначено механізмом розширення Schema.org.

3.1.3 Реалізація опису в JSON

У цьому розділі ми розглянемо особливості реалізації метамоделі як документа JSON (Зверніть увагу на рисунок 3.3 для огляду відповідної схеми JSON). JSON опис слугує досягненню двох основних цілей. З одного боку, він

є основою для документації. З іншого боку, він служить описом доступним для зчитування машинними клієнтами. На рисунку 3.3 ключі, призначені для документування сервісу, представлені курсивом
(name, description, additional_doc, та example), і вони розглядаються як додаткові. Як видно з рисунку, семантична зв'язок між параметрами(**Parameters**), ресурсами(**Resources**) і методами(**Methods**) моделюється ключами *reference* (синього кольору) спеціальні модифікатори для вихідних параметрів представлені зеленим кольором.

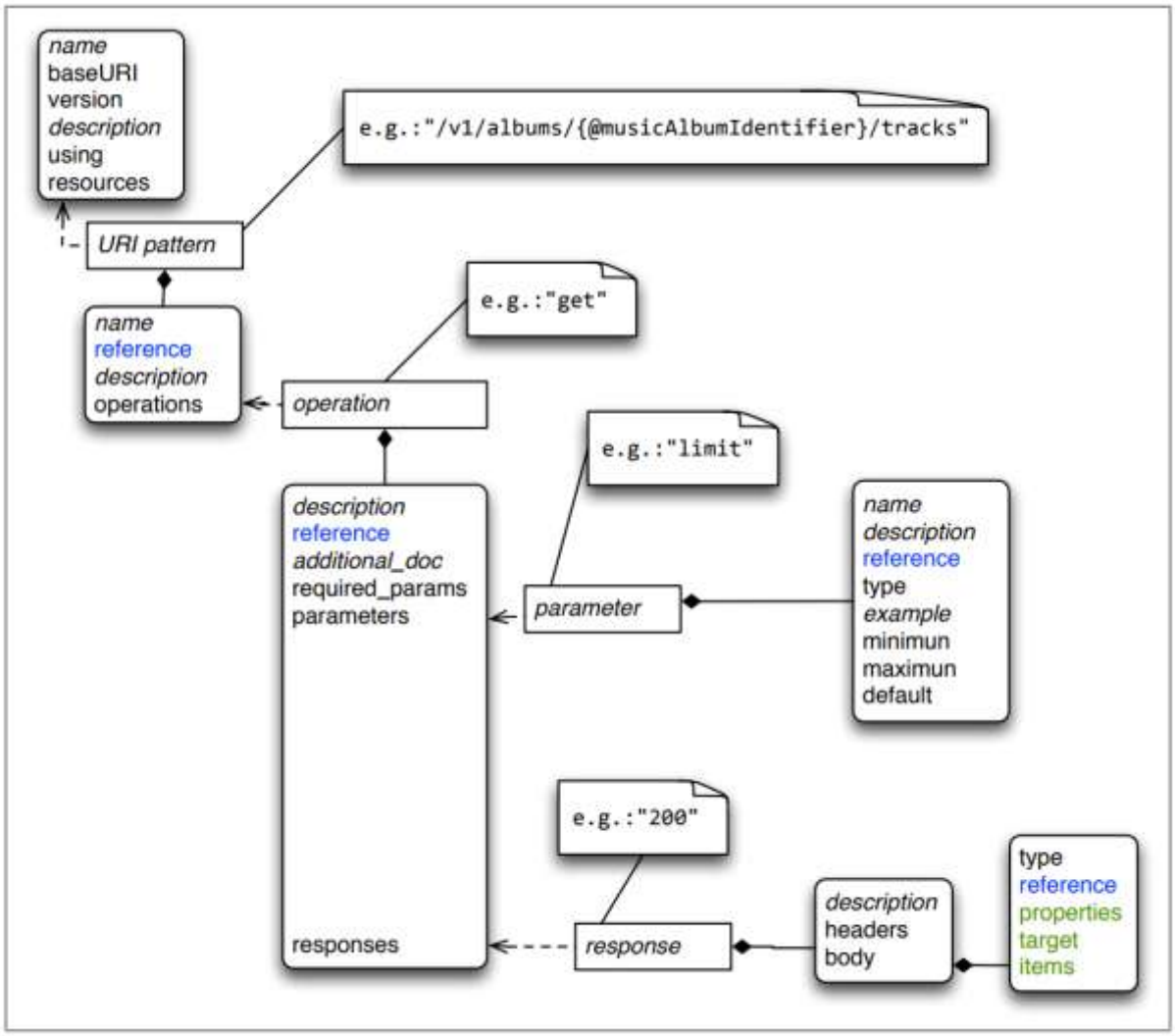

Рисунок 3.3 - Реалізація опису в JSON

В описі JSON повинні бути присутніми такі ключі, як: *baseURI, version, using resources.* Ключ *baseURI* відноситься до кореню вхідних точок сервісу ключ *version* в свою чергу є ідентифікатором для модифікацій в описі, ключ *using* визначає необхідний семантичний словник і на останок ключ *resources* описує кожен ресурс веб-сервісу. Об'єкт ресурсу ідентифікується його URI, який використовується в якості ключа об'єкта. Можна включити семантичні посилання на змінні URI, використовуючи Модифікатор «@», як показано на рисунку 3.3 (/v1/albums /{@musicAlbumIdentifier}), де @musicAlbumIdentifier вказує, що змінна частина URI пов'язана з концепцією, визначеною семантичним словником описаним вище.

Як видно з рисунку 3.4, об'єкт *resource* повинен мати ключі *reference* і *method*. Ключ *reference* пов'язує ресурс з концепцією в словнику. У свою чергу, ключ *method* відноситься до методу мережевого протоколу, доступному для обраного ресурсу (наприклад, GET, POST).

Рисунок 3.4 - Spotify Web API з семантичним описом: *resources* ключовий фрагмент

Ключами для методу виступають *reference, required_params, parameters* та *responses* (рисунок 3.5). Доречно буде з’ясувати яка роль кожного з них. Ключ *reference* пов'язує метод з відповідним семантичним елементом, як описано в словнику. *required_params* пояснює логічні вирази для оцінки

параметрів (при цьому завжди потрібні змінні URI). Об'єкт *parameters* являє собою набір описів, як необхідних так і необов'язкових вхідних параметрів, необхідних для виконайте методу. Параметри можуть бути частиною URI, тіла (потрібно префікс '#') або заголовока (потрібно префікс '!'). Об'єктом *responses* є набір можливих відповідей, семантика яких стосується заданого API. Необов'язковми атрибутами методу виступають *description* і *additional_doc* які знаходять своє застосування у створенні документації.

```
"operations": {
  "get": {
    …
  },
  "post": {
    "description": "Add one or more tracks to a user's playlist.",
    "reference": "@AddAction",
    "additional_doc": "https://developer.spotify.com/web-api/add-tracks-to-playlist/",
    "required_params": "!Authorization AND ((#uris AND !Content-Type) OR NOT #uris)",
    "parameters": {
      "!Authorization": {
        "name": "Authorization Access Token",
        "description": "A valid access token from the Spotify Accounts service.",
        "reference": "@webApplicationApiKey",
        "type": "string"
      },
      "!Content-Type": {
        "name": "Content Type",
        "description": "Required if the IDs are passed in the request body.",
        "reference": "@webApplicationContentType",
        "example": "application/json",
        "type": "string"
      },
      "uris": {
        "name": "Uris",
        "description": "A comma-separated list of Spotify track URIs to add.",
        "reference": "@musicRecordingCollectionUri",
        "type": "string",
        "example": "spotify:track:4iV5W9uYEdYUVa79Axb7Rh"
      },
      "position": {
        "name": "Playlist ID",
        "description": "The position to insert the tracks, a zero-based index.",
        "reference": "@musicPlaylistPosition",
        "type": "integer",
        "example": "2"
      },
      "#uris": {
        "name": "Uris",
        "description": "A JSON array of the Spotify track URIs to add.",
        "reference": "@musicRecordingCollectionUri",
        "type": "array",
        "example": [
          "spotify:track:4iV5W9uYEdYUVa79Axb7Rh",
          "spotify:track:1301WleyT98MSxVHPZCA6M"
        ]
      }
    },
    "responses": {
```

Рисунок 3.5 - Spotify Web API з семантичним описом: *operation і parameter* ключовий фрагмент

До слова, значення *additional_doc* надає посилання, за яким можна знайти додаткову інформацію про метод.

Необхідними ключами для об'єкта *parameter* є *reference* і *type*. Аналогічно попереднім випадкам, ключ *reference* пов'язує параметр з певною семантичною концепцією в словнику. Досить однозначим є і завдання, що виконує *type*, а саме визначення типу змінних. Можливими значеннями є: *string, integer, boolean* і *array*. Додаткові атрибути - це *name, description* та *example* використовуються виключно для цілей документуваня. Значення параметрів також можуть бути обмежені наступними ключами: *enum* (вказання обмеженого набору значень), *default* (значення за замовчуванням), *minimum* вказує мінімальне значення значення для цілих чисел, *maximum* - максимальне. Об'єкт *response* ідентифікується, кодом відповіді для випадку з HTTP.

```
"responses": {
  "200": {
    "description": "On success, the HTTP status code in the response header is 200 OK.",
    "headers": [],
    "body": {
      "type": "object",
      "reference": "@MusicAlbum",
      "properties": {
        "album_type": {
          "type": "string",
          "reference": "@musicAlbumType"
        },
        "artists": {
          "type": "array",
          "reference": "@MusicGroupCollection",
          "items": {
            "type": "object",
            "reference": "@MusicGroup",
            "properties": {
              "href": {
                "type": "hyperlink",
                "target": "/v1/artists/{@musicGroupIdentifier}"
              },
              "id": {
                "type": "string",
                "reference": "@musicGroupIdentifier"
              },
              "name": {
                "type": "string",
                "reference": "@musicGroupName"
              },
```

Рисунок 3.6 - Spotify Web API з семантичним описом: *response* ключовий фрагмент

Необхідними ключами в цьому випадку є е *headers* і *body* (рисунок 3.6). Ключ *headers* являє собою масив що містить відповідні заголовки повідомлень, очікувані у відповіді (наприклад, HTTP-заголовки).

Значення *body* описує наші очікування щодо відповіді. Як видно з вказаного рисунку, *body* має три обов'язкові ключі, а саме: *reference*, який пов'язує представлення ресурсів з концепцією в словнику; *media*, який задає тип відповіді; *type,* який визначає тип даних, що містяться в тілі. У поточній версії ми підтримуємо тільки тип *application/json*. Прийнятні значення для *type* - це ті, які визначені JSON Schema (до прикладу: *string, integer, number, object, array, boolean, null*) а також *hiperlink*. Для значення гіперпосилання нам необхідний ще один ключ *target* для визначення URI ресурсу на який посилається відповідь.

3.1.4 Графове представлення моделі

Було прийнято рішення використовувати графову базу даних для збереження описів, адже відношення між елементами опису формують граф довільної топології. Рисунок 3.7 представляє графову модель, яка втілює метамодель зображену на рисунку 3.1.

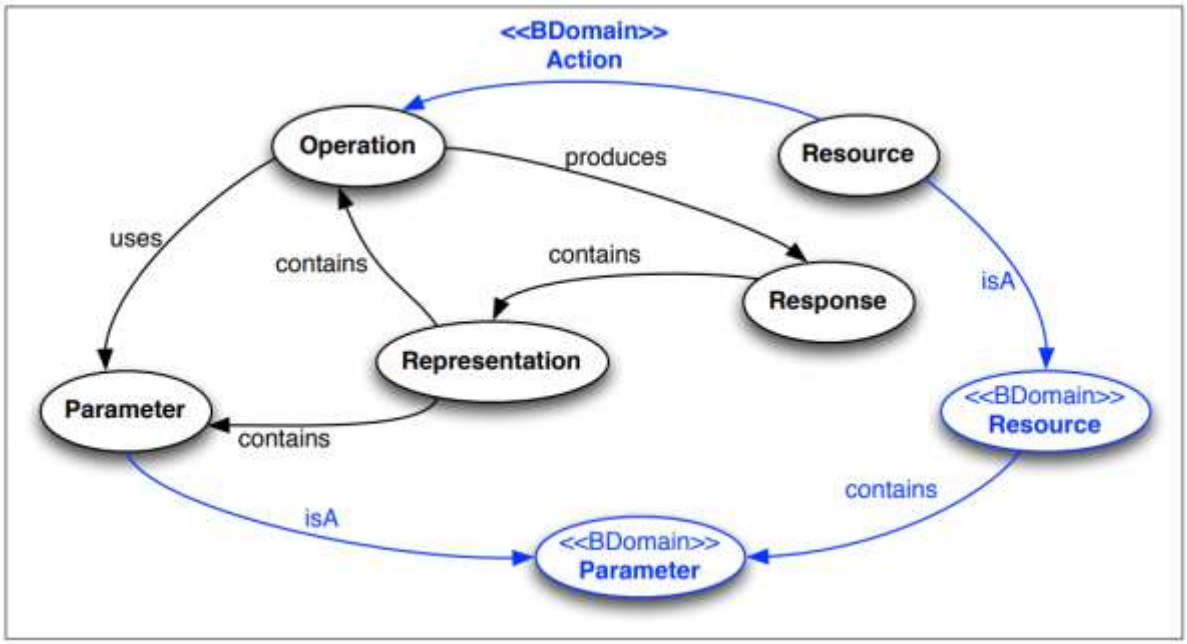

Рисунок 3.7 - Графова модель

3.2 Оркестрування сервісів

Ми розглядаємо оркестрування REST-сервісів, як потік робіт (або шлях в графі), розуміючи під цим набір методів, що дозволяють клієнту досягнути мети, тобто кінцевого стану. Дана мета включає набір вихідних параметрів, отриманих в процесі проходження шляху виконання методів. Методи, у свою чергу виконуються слідуючи якимось обумовленим шаблонам управління. У даній роботі розглядаються тільки три шаблони управління, а саме послідовність, альтернативна та паралельна спліт-синхронізація.

Шаблон послідовності визначає послідовний виклик методів без будь-якої пов'язаної умови. Послідовність може бути виведена із залежності методу від вхідних та вихідних параметрів.

Альтернативний шаблон дозволяє виконувати тільки один з двох можливих сервісів в залежності, в нашому випадку, від рішення користувача. Наш підхід полягає в створенні робочого процесу який включає всі можливі варіанти (методи), але дозволяє користувачеві вибирати. Зауважимо, що

альтернативний шаблон може бути виведений з сигнатури методу (тобто, якщо два методи повертають однаковий результат, але різняться вхідними даним, то вони можуть вказувати альтернативний шаблон).

Input: *input_concepts*, *output_concepts*
Output: *solution_paths*
1: *goal_operations* ← All operations that return at least one concept of *output_concepts*
2: *final_steps* ← Steps containing combinations of *goal_operations*, that executed in parallel return all *output_concepts* (excluding redundant operations for each concept)
3: *banned_steps* ← *final_steps* (or super-sets of each one) banned from being used inside a path
4: **for** *step* in *final_steps* **do**
5: *step.required_param_concepts* ← All the combinations of parameters concepts required to execute all the operations of *step*
6: *path* ← Create path from *step*
7: **for** *combination* in *step.required_param_concepts* **do**
8: **if** *combination* is subset of *input_concepts* **then**
9: Add *path* to *solution_paths*
10: break
11: **end if**
12: **end for**
13: **if** *path* not in *solution_paths* **then**
14: Add *path* to *candidates*
15: **end if**
16: **end for**

Рисунок 3.8 - Псевдокод алгоритму компонування: частина 1

Паралельне поділ дозволяє розділити один потік виконання на дві або кілька гілок. Для шаблону синхронізації потрібно, щоб потік виконання був зупинений до тих пір, поки він не отримає всі результати попередніх методів, які виконувалися паралельно. Два або більше методи, які незалежні по

відношенню один до одного та мають доступ до всіх своїх вхідних параметрів, розглядаються як шаблон з паралельним поділом.

Автоматична оркестрація сервісів реалізується алгоритмом зворотного відстеження (рисунок 3.8 і 3.9). Для управління неоднорідністю даних ми розглядаємо семантичний еквівалент вхідних і вихідних параметрів, замість кожного конкретного. Входом алгоритму(*Input*) представлений клієнтським запитом(*composition_request*), який визначає необхідну мету: набір понять в словнику, пов'язаних з вихідними параметрами (*output_concepts*). При бажанні, клієнт може вказувати набір концепцій, пов'язаних з вхідними параметри (*input_concepts*), які представляють інформацію, яку клієнт має на вході. Паралельні та альтернативні шляхи можуть бути зведені до моделі послідовності, щоб обчислити критичний шлях виконання. Ми розглядаємо таку критичну послідовність, як кроки (шлях - послідовність кроків). Крок - це набір операцій, що не залежать між собою і їх вихідні параметри потрібні наступним крокам на шляху виконання. Операції на останньому етапі(*final_step*) повинні повертати всі результуючі що потрібно. Операції на першому етапі(*first_step*) повинні виконуватися з використанням даних, що надав клієнт (при необхідності). Протягом шляху виконання, кожна операція повинна виконуватися не більше одного разу.

Алгоритм починається з визначення операцій (*goal_operations*), які породжують хоча б одну концепцію (*output_concept*), як видно з рядку 1. Потім набір можливих заключних кроків (*final_steps*) створюється з *goal_operations* (рядок 2). Варто розглядати *final_steps, як* комбінацію операцій (або однієї операції) який генерує всі концепції виведення. Операція включається в крок, якщо вона повертає параметр, який очікується на даному кроці, враховуючи, що ніяка інша операція цього кроку не повертається бажану концепцію. Потім для кожної операції на кожному *final_step* алгоритм знаходить всі комбінації необхідних вхідних понять, необхідних для виконання кроку (рядок 5).

Враховується ситуація, коли конкретна комбінація включає додаткові параметри, крім наданих клієнтом. У цьому випадку алгоритм визначає, що такий крок вимагає додаткового аналізу та позначає крок, кандидатом на виконання (*candidate),* про що говорить нам рядок 14. Кандидати - це шляхи, які містять операції, що повертають всі необхідні результати, але вимагають більше вхідних даних ніж ті, що визначені клієнтом. Кандидати розглядаються в зворотному порядку. Оскільки процес розгляду всіх кандидатів може споживати велику кількість ресурсів і часу, часто необхідно встановлювати межі процесу (наприклад, час очікування, максимальна кількість кроків, максимальна кількість операції, максимальна кількість рішень тощо). Якщо відомі всі вхідні параметри кроку, і як наслідок, кожна операція кроку може бути повністю виконана, шлях, до якого належить крок, вважається пріоритетним для вирішення завдання вирішення (рядок 9).

Потім оцінюються всі кандидати (рядок 17). Щоб оцінити кандидата(*candidate*), алгоритм ідентифікує *first_step* на шляху кандидата (тобто новий крок) і комбінацію всіх вхідних параметрів для виконання операцій на даному кроці (рядок 20-22). Потім алгоритм знаходить можливі *previous_steps* (рядок 23), які повертають необхідні дані для виконання *first_step*.

```
17: while candidates do
18:   for path in candidates do
19:     Remove path from candidates
20:     for combination in path.first_step.required_param_concepts do
21:       previous_operations ← All operations that return a parameter concept from
    combination, not available in input_concepts
22:     end for
23:     previous_steps ← Steps containing combinations of previous_operations, that
    executed in parallel return all parameter concepts in combination (excluding redun-
    dant operations for each concept and banned operations of the path), and different to
    banned_steps
24:     for step in previous_steps do
25:       step.required_param_concepts ← All the combinations of parameters con-
    cepts required to execute all the operations of step
26:       new_path ← Create new path from path and prepend step to it
27:       for combination in step.required_param_concepts do
28:         if combination is subset of input_concepts then
29:           Add new_path to solution_paths
30:           break
31:         end if
32:       end for
33:       if new_path not in solution_paths then
34:         Add new_path to candidates
35:       end if
36:     end for
37:   end for
38: end while
return solution_paths
```

Рисунок 3.9 - Псевдокод алгоритму компонування: частина 2

Наприклад, припустимо, що для виконання операції на першому кроці потрібні наступні вхідні параметри A, B і C. Алгоритм знаходить попередні

операції *Op1 (повертає {A}), Op2 (повертає {B, C}) і Op3 (повертає {A, C})*. Отже, попередніми кроками були б *Step1 (Op1 і Op2) і Step2 (Op2 і Op3)*. Зверніть увагу, що Op1 і Op3 не були б частиною одного і того ж кроку через надмірність концепції A та відсутність параметру *B*. Однак *Op2 і Op3* є частиною *Step2*, навіть якщо існує надмірність концепції C, оскільки кожна операція забезпечує всі необхідні концепції. Як говорилося раніше, операції всередині кроку можуть виконуватися паралельно (вони не залежать від них). Кожен крок шляху виконується перед наступними кроками (шаблон послідовності).

Щоб визначити, чи є кандидат (*candidate*) шуканим рішенням, алгоритм оцінює, чи операції на попередньому кроці мають всі вхідні параметри(рядок 28). У такому випадку попередній крок додається першим кроком до кандидата, який тепер вважається рішенням (рядок 29). Якщо попередній крок не може бути виконаний, новий шлях розглядається як кандидат, і оцінка кандидатів триває. Оцінка кандидатів припиняється коли більше немає кандидатів, або весь граф пройдено, і немає більше операцій, які можуть бути додані до шляху.

```
def get_solution_path(input_concepts):


    for step in final_steps:
        params = get_required_paameters(step)
        path = create_path_from_step(step)

        for combination in params:
            if combination.issubset(input_concepts):
                solution_path.append(path)

        if path not in solution_path:
            candidates.append(path)

    while candidates:
       for path in candidates:
           candidates.remove(path)

           for combination in path.first_step.required_parameters:
               previous_operations = get_previous_operations(combination, input_concepts)

           previous_steps = get_previous_steps(previous_operations)
           for step in previous_steps:
               params = get_required_paameters(step)
               new_path = generate_new_path(path, step, params)

               for combination in step.required_params:
                   if combination.issubset(input_concepts):
                       solution_path.append(new_path)

               if new_path not in solution_path:
                    candidates.append(new_path)

    return solution_path
```

Рисунок 3.10 - Фрагмент реалізації описаного алгоритму мовою Python

Складність алгоритму залежить в основному від кількості операцій, яку повертає *output_concept*. Крім того, на нього впливає кількість необхідних параметрів для кожного етапу.

Дальше алгоритм роботи програми зводиться до пошуку оптимального варіанту композиції на основі наявних даних. Для кращого розуміння підходу наведемо загальний вигляд онтології сервісу, з якою ми працюємо(Рисунок 3.11). Додамо, що онтології у нас зберігатимуться в графовій базі даних, а не в базі даних для збереження триплетів. Завдяки цьому, ми можемо оглянути представлення описаної онтології в зручному форматі(Рисунок 3.12).

```
//CLASSES (Neo4j's Labels)
(service_class:Class {  uri:"http://semantic.com/voc/services#Service",
                        label:"Service",
                        comment:"Microservice that performs some funtionality"}),

//DATATYPE PROPERTIES (Neo4j's node properties)
(name_dtp:DatatypeProperty {    uri:"http://semantic.com/voc/services#name",
                                label:"name",
                                comment :"Microservice name"}),
(name_dtp)-[:DOMAIN]->(service_class),

(url_adress_dtp:DatatypeProperty {  uri:"http://semantic.com/voc/services#url",
                                    label:"url",
                                    comment :"Url adress to comminicate with service "}),
(url_adress_dtp)-[:DOMAIN]->(service_class),

(description_dtp:DatatypeProperty {  uri:"http://semantic.com/voc/services#description",
                                     label:"description",
                                     comment :"Description of service functionality "}),
(description_dtp)-[:DOMAIN]->(service_class),

(inputs_dtp:DatatypeProperty {  uri:"http://semantic.com/voc/services#inputs",
                                label:"inputs",
                                comment :"Input format our service is waiting for "}),
(inputs_dtp)-[:DOMAIN]->(service_class),

(outputs_dtp:DatatypeProperty {  uri:"http://semantic.com/voc/services#outputs",
                                 label:"outputs",
                                 comment :"Output format service will return "}),
(outputs_dtp)-[:DOMAIN]->(service_class),

(pre_conditions_dtp:DatatypeProperty {  uri:"http://semantic.com/voc/services#pre_conditions",
                                        label:"pre-conditions",
                                        comment :"Conditions on the inputs of the service defined by a given domain ontology "}),
(pre_conditions_dtp)-[:DOMAIN]->(service_class),

(post_conditions_dtp:DatatypeProperty { uri:"http://semantic.com/voc/services#post_conditions",
                                        label:"post-conditions",
                                        comment :"Conditions on the outputs of the service defined by a given domain ontology"}),
(post_conditions_dtp)-[:DOMAIN]->(service_class)
```

Рисунок 3.11 - Приклад онтології описаних сервісів

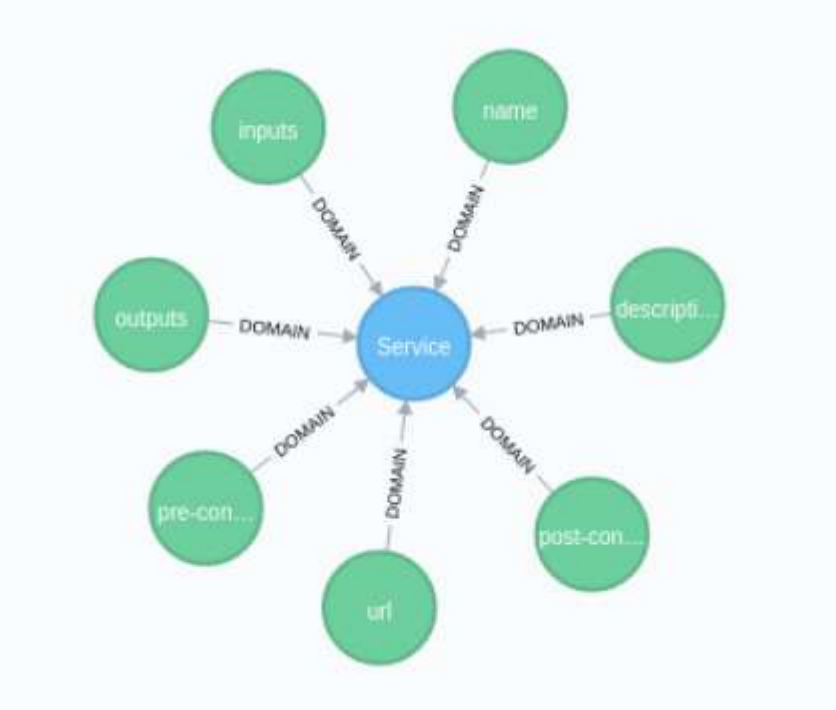

Рисунок 3.12 - Графічне представлення онтології

Алгоритм оптимізації результатів компонування сервісів заключається у визначенні найбільш доречної композиції з урахуванням вимог користувача.

Для цієї мети була написати функція пост-фільтрації отриманих графів з відповідною пріоритезацією їх, як рішення завдання. Для отримання шуканих результатів ми використовуємо безпосередньо онтологію сервісів, а саме вхідні та вихідні параметри та опис. Користувачу надається можливість задати як вхідні параметри, так і очікувані результати. Це дає змогу нам обирати максимально відповідні сервіси для кожного випадку. Маючи такі результати, алгоритм оцінює отриманий набір графів(шляхів) виходячи з ваги сервісів, кількості кроків, наявності необхідних даних для їх виконання тощо.

3.3 Висновки

Підсумовуючи результати отримані в даному розділі, згадаємо основні підходи що були запропоновані. Перш за все, була розроблена унікальна метамодель взаємодії сервісів, яка передбачає два рівні взаємодії: семантичний та активності. Для реалізації семантичного рівня, було розширено словник Schema.org та розширено JSON розробленими описами. Механізм додавання нових елементів у формат був детально висвітлений низкою прикладів з обширним поясненням. У другій частині розділу, ми маємо змогу ознайомитися з алгоритмом компонування сервісів. на основі побудови графу виконання операцій. Пояснення було проілюстовано, як псевдокодом з покроковим поясненням дій, так і власне фрагментом реалізації основної логіки мовою Python.

4 РЕАЛІЗАЦІЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

4.1 Архітектура додатку та вибір технологій

Відштовхуючись від основного завдання, що виносилось у даній роботі, а саме розробка алгоритму компонування REST-сервісів на основі їх семантики, було розроблено прототип.

Для створення прототипу, який би надав можливість перевірити запропоновану концепцію, було проглянуто цілу низку Web API. Основними критеріями відбору сервісів були: дотримання обмежень REST, надання вичерпної документації, потрапляння у відповідну предметну область. Незважаючи на те, що деякі обмеження задовольнялись не в повному обсязі, ми вибрали наступні три веб-сервісів: Spotify, Songkick і Uber. Spotify API надає доступ до каталогу служби потокової передачі музики. Songkick API надає доступ до музичної базу даних з інформацією про майбутні і минулі концертах, а також сетлісти. Uber API дозволяє клієнту запитувати типи транспортних послуг, порівнювати ціни і час прибуття, а також надає профіль користувача і інформацію про його діяльність.

Зауважимо, що JSON-опис для кожного Web API було створено вручну, як і словник заснований на Schema.org. Нам довелося розширити словник для моделювання концепцій розглянутих в Web API. Наш підхід був реалізований шляхом написання парсера на Python, який перетворює наш JSON опис та словникові файли в граф. Файли JSON перевіряються за допомогою JSON Schema перед парсингом. Характерно, що ми обрали популярну базу даних Neo4j для зберігання набору даних, оскільки вона надає власну модель графа. Крім того, за допомогою бібліотеки Py2Neo, як завантаження, так і взаємодія з даними відчутно спрощується. Алгоритм компонування був реалізований також мовою Python і використовує бібліотеку Py2Neo для доступу до бази даних.

Архітектура нашого додатку представлена на рисунку 4.1. Доступ до бази даних здійснюється модулю *core module* написаному мовою Python3. Два інші модулі написані цією ж мовою це *vocabulary-parser* та *json-description-parser.* Перший відповідає за парсинг нашого словника та додавання семантичного рівня до графу. В той час, як *json-description-parser* обробляє JSON документи і додає сервіси до графу створюючи тим самим рівень активності.

Важливу роль відіграє модуль *query-engine* з допомогою якого ми маємо змогу надсилати запити до сформованого графу. Саме він допомагає нам обрати необхідний запит грунтуючись на запиті, написаному природною мовою та сформованим в результаті параметрах. Даний модуль використовує бібліотеку Natural Language Toolkit's (NLTK) для опрацювання запитів природною мовою. А саме використовуються алгоритм *PorterStemmer,* та функції *pos_tag* і *word_tokenize.* Після того, як фразу було проаналізовано, та виділено основні концепції та дії кожна з них порівнюється з існуючою семантикою графу використовуючи сервіс *UMBC Semantic Similarity.*

Модуль ж який відповідає за безпосередню взаємодію з користувачем, тобто *web-app* написаний з використанням фреймворків Flask та React.js.

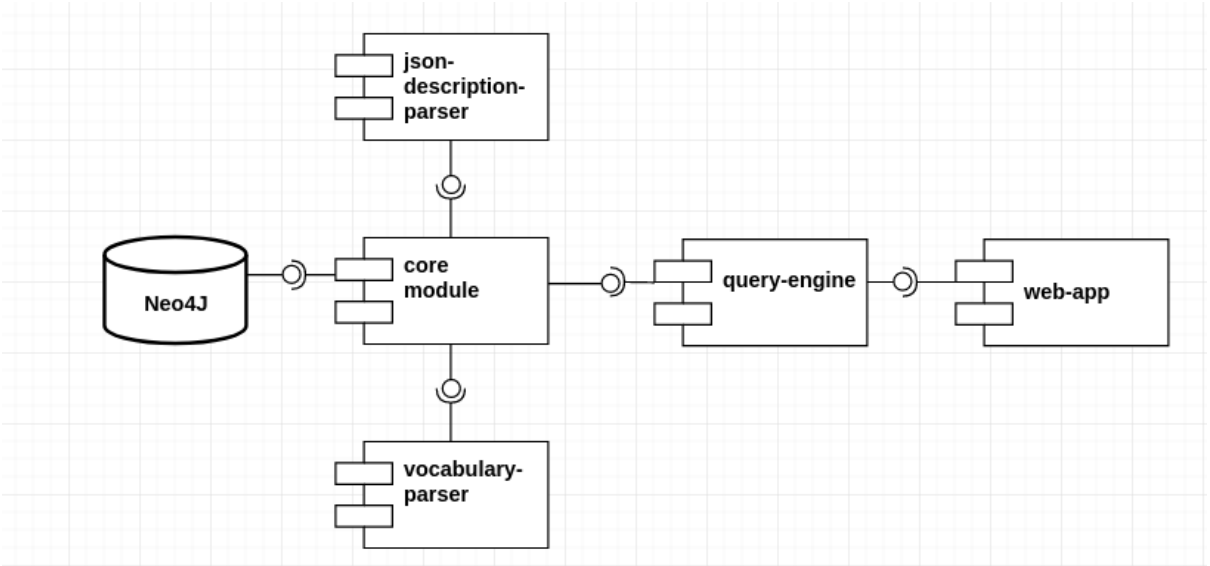

Рисунок 4.1 - Діаграма компонентів додатку

Вельми цікавим елементом додатку є механізм спілкування з базою даних, для отримання всієї необхідної інформації стосовно семантики сервісів.

(a)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters)
WHERE concept.GRI=~ 'http://schema.org/Thing/CreativeWork/MusicPlaylist.*'
AND action.GRI = 'http://schema.org/Action/GetAction/GetMusicRecordingCollectionAction'
AND concept: `Resource Concept`  AND resources: Resource  AND operations: Operation
AND parameters: Parameter
RETURN concept, resources, operations, parameters
```

(b)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/MusicGroupCollection'
AND action.GRI = 'http://schema.org/Action/GetAction/GetMusicGroupCollectionAction/GetSimilarArtistsAction'
AND concept: `Resource Concept`  AND resources: Resource  AND operations: Operation
AND parameters: Parameter  AND responses: Response
RETURN concept, resources, operations, responses, parameters
```

(c)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/MusicEventCollection'
AND action.GRI = 'http://schema.org/Action/SearchAction/MusicEventSearchAction'
AND concept: `Resource Concept`  AND resources: Resource  AND operations: Operation
AND parameters: Parameter  AND responses: Response
RETURN concept, resources, operations, responses, parameters
```

(d)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/TaxiCollection'
AND action.GRI = 'http://schema.org/Action/GetAction/GetTaxiCollectionAction'
AND concept: `Resource Concept`  AND resources: Resource  AND operations: Operation
AND parameters: Parameter  AND responses: Response
RETURN concept, resources, operations, responses, parameters
```

(e)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/TaxiFareCollection'
AND action.GRI = 'http://schema.org/Action/GetAction/GetTaxiFareCollectionAction'
AND concept: `Resource Concept`  AND resources: Resource  AND operations: Operation
AND parameters: Parameter  AND responses:Response
RETURN concept, resources, operations, responses, parameters
```

(f)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/TaxiTravelTimeCollection'
AND action.GRI = 'http://schema.org/Action/GetAction/GetTaxiTravelTimeCollectionAction'
AND concept: `Resource Concept`  AND resources: Resource  AND operations: Operation
AND parameters: Parameter  AND responses:Response
RETURN concept, resources, operations, responses, parameters
```

Рисунок 4.2 - Запити Cypher

Давайте звернемо увагу на Рисунок 4.2. На ньому наведено кілька характерних запитів до бази даних, що виконувалися в процесі роботи. Зауважимо, що вони зроблені декларативною мовою запитів Cypher. Результати виконання деяких із них зображені на рисунку 4.3.

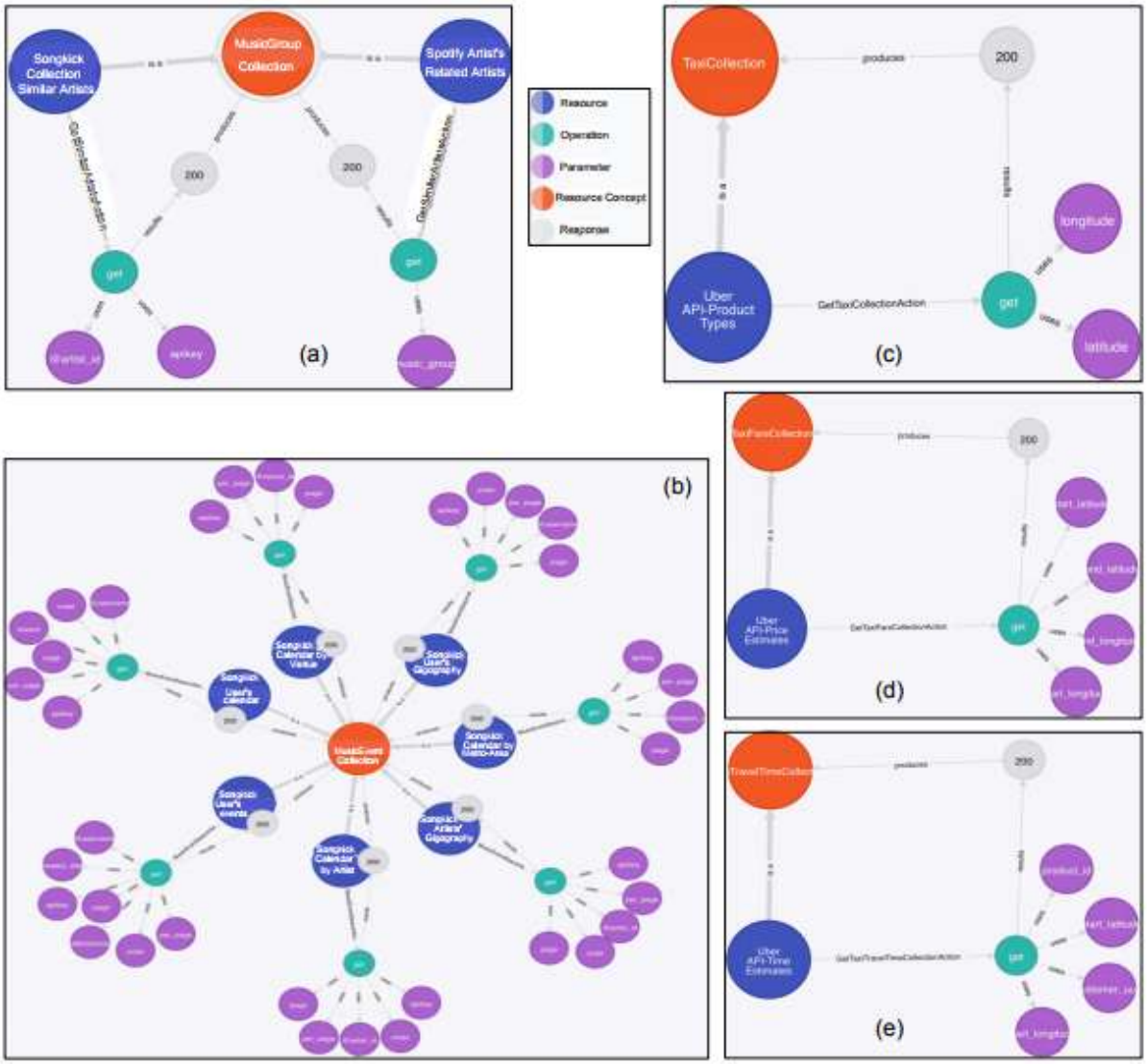

Рисунок 4.3 - Результати виконання запитів

Наступні кілька сторінок хочеться посвятити власне інтерфейсу прикладного рішення. прикладам взаємодії з куристувачем та відоповідями на запити природною мовою. Розпочнемо з рисунку 4.4, який показує головну сторінку.

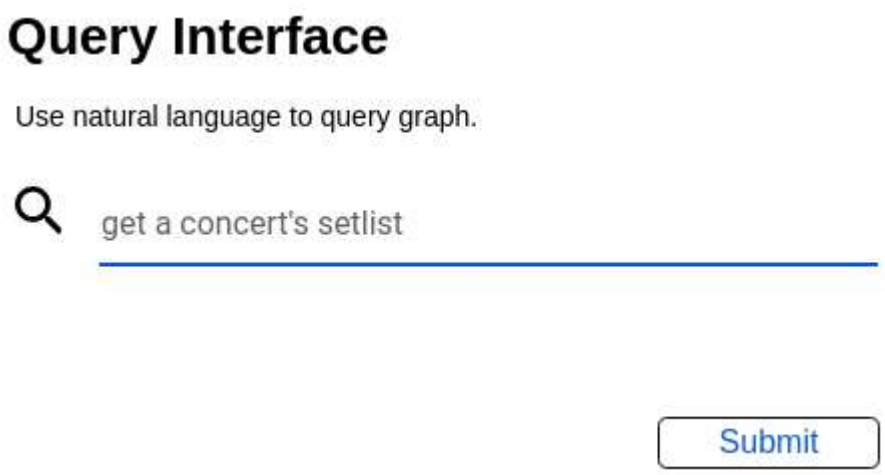

Рисунок 4.4 - Інтерфейс веб-додатку: створення запиту

Далі додамо наглядності й продемонструємо можливі варіанти відповіді на запити. На рисунках 4.5-4.6 зображено відповідь на запити у формі таблиці. В той час, як на рисунку 4.7, ми можемо спостерігати граф. Нижче графу розташований детальний опис відповіді.

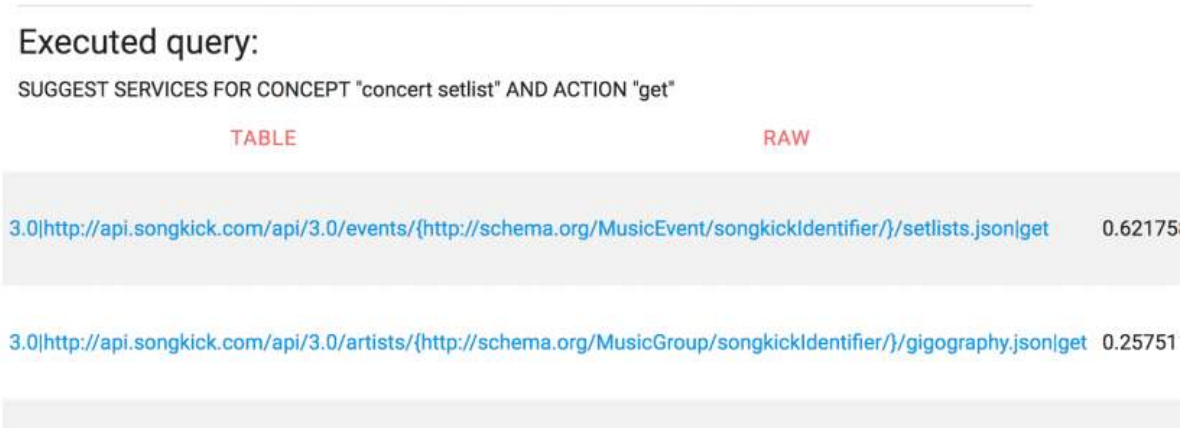

Рисунок 4.5 - Інтерфейс веб-додатку: відповідь на запит

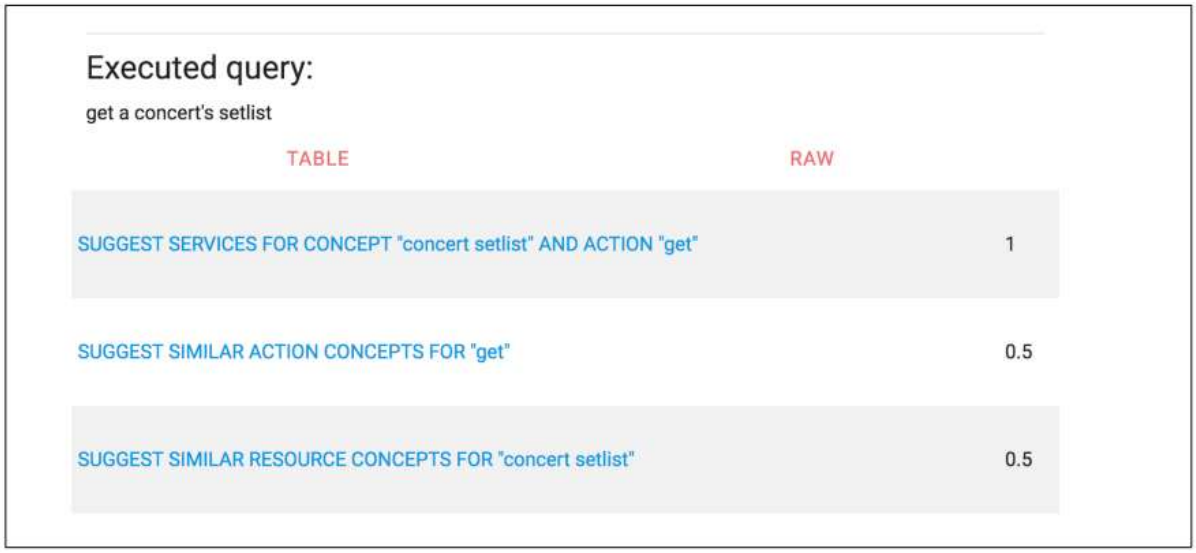

Рисунок 4.6 - Інтерфейс веб-додатку: пропоновані результати

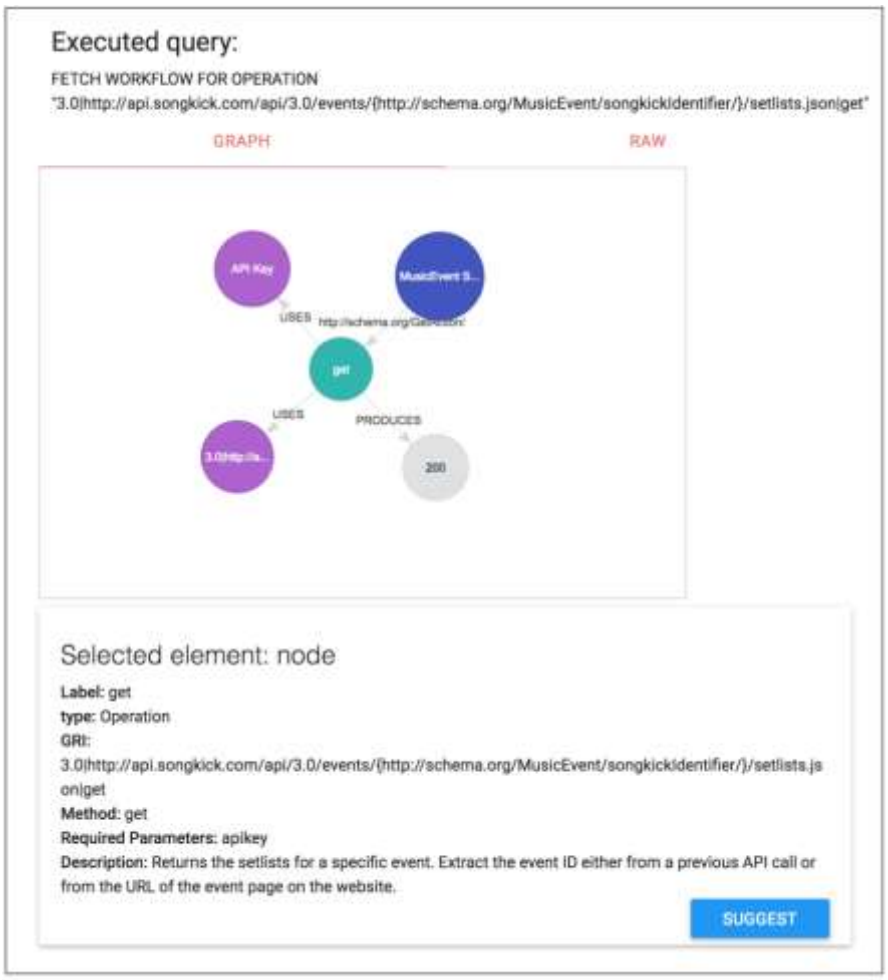

Рисунок 4.7 - Інтерфейс веб-додатку: графова відповідь

Далі наведемо дані стосовно ребер та вершин сформованого графу для кожного з сервісів (Таблиця 4.1).

Таблиця 4.1 - Вершини і ребра в базі даних

Назва сервісу	**Вершини**	**Ребра**
Spotify	611	130
Uber	175	304
Songkick	318	579
Загалом	1277	2202

У наступних двох таблицях наводяться дані про вершини для семантичного шару та шару активності(Таблиці 4.2-4.3).

Таблиця 4.2- Вершини семантичного шару

API	**Resource**	**Actions**	**Parameter Concepts**
Spotify	10	9	63
Uber	7	4	46
Songkick	6	3	39

Таблиця 4.3- Вершини шару активності

API	**Resources**	**Operations**	**Parameters**	**Responses**	**Representa-tions**
Spotify	20	27	513	28	23
Uber	10	11	129	15	10
Songkick	15	15	260	15	13

4.2 Постановка завдання та оцінка результатів

Розглянемо прикладну задачу композиції сервісів: *Користувач хоче відвідати концерт його улюбленого гурту але найближчим часом концертів не передбачається. Тому він вирішує сходити на концерт схожого виконавця. Для цього йому потрібно дізнатися конкретне місце проведення, а також деталі поїздки на таксі на обрану подію.*

Така ціль може бути досягнута операціями GET (інформація про виконавців, концертні дані, ціни тощо) до ресурсів Songkick або Spotify. Взаємодія з Uber потрібна консультацій стосовно тарифів (GET), а потім для виклику таксі. Характерно, що інтерфейси неоднорідні і семантика сервісу дозволить виявити відповідні ресурси. Крім того, необхідно опрацьовувати різні ситуації (наприклад, гурт не підходить по стилю, місце події може бути занадто далеко, або оплата за таксі може бути занадто високою). Також важливо розглянути семантику операцій, оскільки ресурс може включати кілька посилань, що відповідають різним запитам з різною семантикою.

Для описаного сценарію розглянемо три випадки вхідних параметрів(*input_concepts*):

- 1 параметр: єдиним наданим параметром буде обов'язковий API ключ (*http://schema.org/WebApplication/apiKey/*)
- 2 параметри: до попереднього параметру додається ідентифікатор музичного гурту (http://schema.org/MusicGroup/identifier/).
- 3 параметри: клієнт не знає вхідних параметрів але знає як знайти їх використовуючи пошукові можливості API. Параметри, що стосуються пошуку (http://schema.org/Search/type/ і http://schema.org/Search/query/) вважатимемо вхідними.

Наші досліди проводитимемо на процесорі Intel Core i7-7700 3.6GHz з оперативною пам'яттю об'ємом 8 GB. Операційна система Ubuntu 14.04. Було здійснено 10 замірів які потім усереднили для отримання точнішого результату. Результати роботи алгоритму наведемо для всіх трьох випадків з відповідною кількістю вхідних параметрів. Кількість кроків алгоритму обмежено 7, щоб виключити надто довге виконання. Для ознайомлення з результатами оцінки описаного сценарію розглянемо таблицю 4.4.

Таблиця 4.4- Кількість розв'язків на кожному кроці

Вхідні параметри	**Кроки**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
http://schema.org/WebApplication/apiKey	0	1	11	132	2	25	336
http://schema.org/WebApplication/apiKey http://schema.org/MusicGroup/identifier/	0	3	20	178	3	18	103
http://schema.org/WebApplication/apiKey http://schema.org/Search/type/ http://schema.org/Search/query/	0	3	22	350	6	78	1

Таблиця 4.5- Підсумкові результати рішення

Вхідні параметри	Всього розв’язків	Залишилось кандидатів	Час виконання (с)
http://schema.org/WebApplication/apiKey/	364	4663	394.1312 ± 11.9393
http://schema.org/WebApplication/apiKey/ http://schema.org/MusicGroup/identifier/	123	343	78.4665 ± 0.9224
http://schema.org/WebApplication/apiKey/ http://schema.org/Search/type/ http://schema.org/Search/query/	1094	5322	288.9120 ± 2.6779

Ми можемо спостерігати велику кількість розв’язків для кожного з випадків. Такі результати можна пояснити, тим що тільки одна операція вирішує результат оцінки поїздки але жоден з необхідних вхідних параметрів не надається в якості вхідних параметрів. Як результат, алгоритм шукає альтернативи цим параметрам. Що відображається на результатах, де видно, що найкоротший шлях розв’язку займає два кроки.

Дальше наведемо знайдені варіанти композиції сервісів для кожного з трьох варіантів вхідних параметрів(Рисунки 4.8-4.10). Зауважимо, що результати наведені для другого та третього кроку.

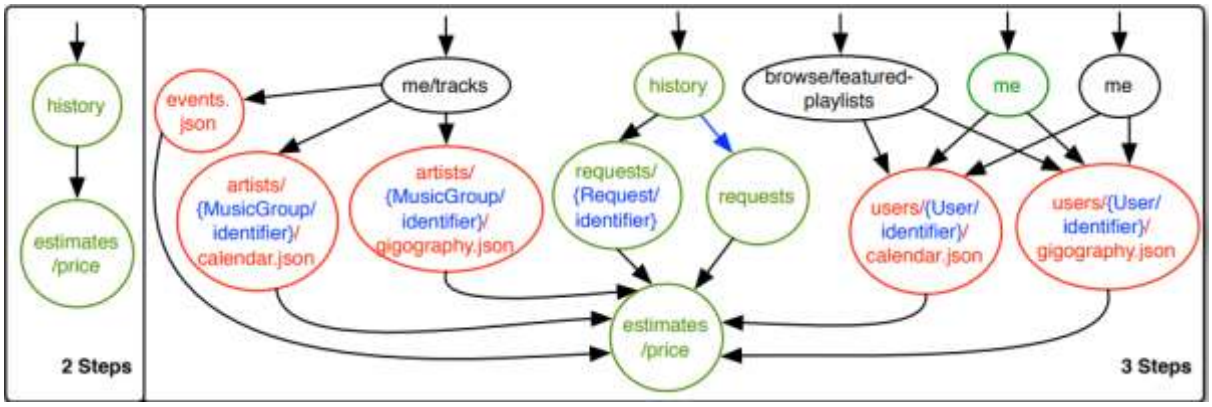

Рисунок 4.8 - Композиція сервісів для заданого сценарію (1 вхідний параметр)

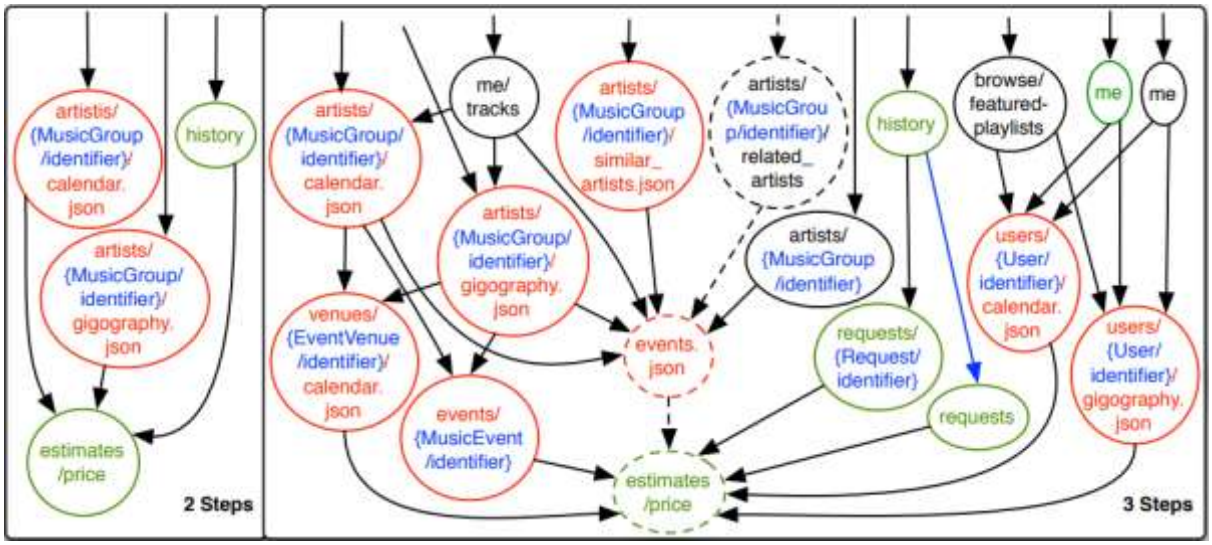

Рисунок 4.9 - Композиція сервісів для заданого сценарію (2 вхідні параметри)

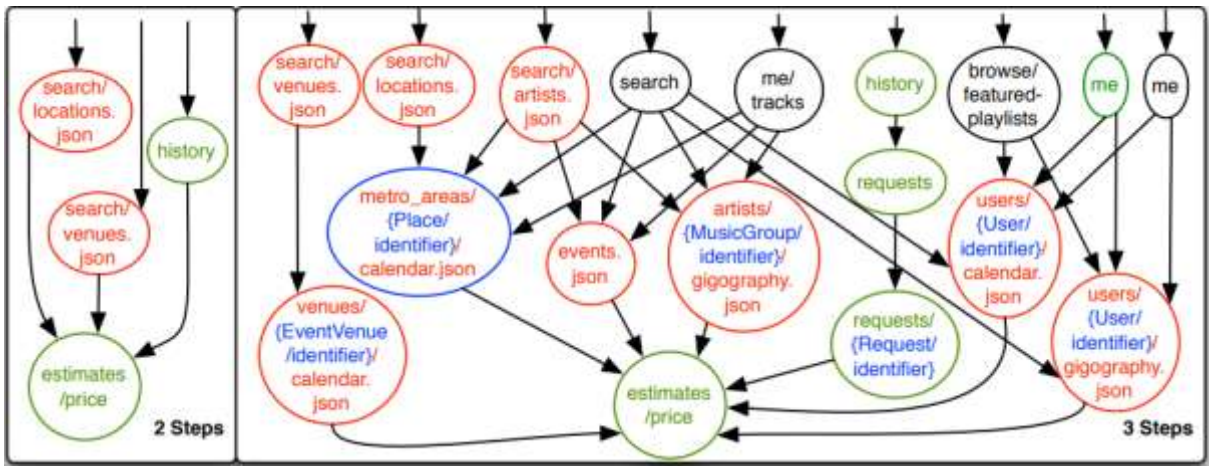

Рисунок 4.10 - Композиція сервісів для заданого сценарію (3 вхідні параметри)

4.3 Висновки

Давайте стисло підведемо підсумок даного розділу, та дамо оцінку отриманим результатам. В його першій частині у розгорнутій формі пояснюється використання обраних технологій. Робиться акцент на основних компонентах створеного додатку, наводиться відповідна діаграма. Разом з тим можна ознайомитися і з інтерфейсом програмного рішення. У другій частині описується прикладний сценарій для оцінки роботи алгоритму. Результати замірів знайшли своє відображення у таблиця та наглядних ілюстраціях.

5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ “СЕМАНТИЧНЕ ОРКЕСТРУВАННЯ REST-СЕРВІСІВ ”

5.1 Опис ідеї стартап-проекту

Розділ має на меті проведення маркетингового аналізу стартап проекту “Семантичне оркестрування REST-сервісів” задля визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження.

Опис стартап-проекту “Семантичне оркестрування REST-сервісів” наведено у Таблиці 4.1.

Таблиця 4.1. Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Створення семантичного опису REST-сервісів та подальше оркестрування на їх основі. Рішення передбачає використання графової бази даних та описів у форматі JSON.	1. Використання для відлагоження взаємодії сервісів на основі семантики.	Вигода для користувача поляга у можливості отримання повної композиції сервісів та, як наслідок кінцевого результату без складних маніпуляцій.
	2. Дослідження сервісів машинними клієнтами.	Використання Neo4J дозволяє користувачу оцінити переваги та недоліки кожного з пропонованих варіантів оркестрування сервісів.

Метою розділу є формування інноваційного мислення, підприємницького духу та формування здатностей щодо оцінювання ринкових перспектив і можливостей комерціалізації основних науково-технічних розробок, сформованих у попередній частині магістерської дисертації у вигляді

розроблення концепції стартап-проекту “Семантичне оркестрування REST-сервісів” в умовах висококонкурентної ринкової економіки глобалізаційних процесів.

Отже, проект “Семантичне оркестрування REST-сервісів” може бути використаним як інструмент для отримання повної композиції сервісів на основі їх семантичного опису, зробивши запит природною мовою. Яка в результаті обробляється програмними засобами і на виході користувачу пропонується низка рішень з відсортована необхідним чином.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	*Техніко-економічні характеристики ідеї*	*(потенційні) товари/концепції конкурентів*				*W (слабка сторона)*	*N (нейтр а- льна сторон а)*	*S (сильна сторона)*
		Мій проект	*Конкурент 1*	*Конкурент2*	*Конкурент 3*			
1.	Форма виконання	Веб-сервіс	Програма	Веб-додаток	Програма			+
2.	Собівартість	Низька	Низька	Висока	Висока			+
3.	Кросплатформність	Так	Ні	Так	Ні		+	
4.	Наявність графової бази даних	Так	Так	Так	Так		+	
5.	Універсальні описи сервісів	Ні	Ні	Ні	Ні		+	
6.	Обробка природної мови	Так	Ні	НІ	НІ			+
7.	Мультиагентність	Ні	Ні	Ні	Так	+		

Сильними сторонами проекту є форма виконання у вигляді веб-сервісу, низька собівартість, використання графової бази даних, що додає відчутну гнучкіть проекту. Слабкою стороною є відсутність універсальних описів, чим

до слова, не пожуть похвалитися і конкуренти. та відсутність мультиагентності. Отож, система є конкурентноспроможною.

5.2 Технологічний аудит ідеї проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару).

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

№ п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1.	База даних	Neo4J	Наявна	Безкоштовна, доступна
2.	Створення REST API для доступу до бази даних	Flask, ReactJS, git, NLTK	Наявні	Безкоштовна, доступна
3.	Хмарне розгортання додатку	Digital Ocean	Наявна	Безкоштовна, доступна
		AWS	Наявна	Платні

Обрані технології реалізації ідеї проекту: Flask через повну безкоштовність фреймворку та наявність докладної документації, наявність досвіду роботи розробників з даною технологією; ReactJS через простоту використання, безкоштовність та можливість розгортання додатків на основі таких технологій у хмарі; Digital Ocean для розгортання у хмарі через безкоштовність.

5.3. Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1.	Кількість головних гравців, од	2
2.	Загальний обсяг продаж, грн/ум.од	10 000 грн./ум.од
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Немає
5.	Специфічні вимоги до стандартизації та сертифікації	Немає
6.	Середня норма рентабельності в галузі (або по ринку), %	R = (4000000 * 100) / (1000000 * 12) = 33%

Отже, було проаналізовано наявність попиту, обсяг, динаміку розвитку ринку. Обмеження для входу на ринок відсутні, динаміка ринку зростає, галузь є рентабельною.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (табл. 5.5).

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1.	Необхідне програмне забезпечення, що би дозволило складати сервіси в композицію на основі семантичних описів	Потенційними цільовими групами є дослідницькі центри, університети та компанії, специфіка роботи яких потребує компонування REST-сервісів	Цільова група займається впровадженням графових баз даних у семантичний веб	Рішення повинне бути придатним до інтеграції в інші більш складні системи, має надавати користувачу інтерфейс для взаємодії та працювати в розумних часових рамках

Згідно проведеної характеристики потенційних клієнтів стартап-проекту випливає, що на ринку є затребуваним програмне забезпечення для оркестрування семантичних сервісів і потенційними цільовими групами є дослідницькі центри, університети та компанії, специфіка роботи яких потребує компонування сервісів на основі семантичних описів.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. № 5.6-5.7). Фактори в таблиці подавати в порядку зменшення значущості.

Таблиця 5.6 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1.	Конкуренція	Вихід на ринок компанії зі значними ресурсами	1. Вихід з ринку 2. Запропонувати великій компанії поглинути себе 3. Передбачити додаткові переваги власного ПЗ для того, щоб повідомити про них саме після виходу міжнародної компанії на ринок
2.	Зміна потреб користувачів	Користувачам необхідне програмне забезпечення з іншим функціоналом	1. Передбачити можливість додавання нового функціоналу до створюваного ПЗ
3.	Зміна тарифів провайдера хмарного розгортання на платні	Необхідність оплати послуг провайдера хмари	1. Пошук іншого безкоштовного провайдера 2. Пошук інвестицій для оплати існуючого провайдера
4.	Надходження на ринок альтернативних продуктів	Перехід користувачів нашого товару на інший продукт	Впровадження нового функціоналу, якого немає у конкурентів
5.	Уповільнення росту ринку	Скорочення користувачів продуктів, що тільки виходять на ринок	Інвестиції у впровадження ефективної реклами продукту

Отже, було проаналізовано фактори загроз ринкового впровадження проекту, серед яких: конкуренція, уповільнення росту ринку, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні та надходження на ринок альтернативних продуктів. Було також запропоновано можливі реакції компанії.

Таблиця 5.7 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1.	Стрімкий ріст попиту на інструменти компонування сервісів	Наявність попиту на інструменти для компонування сервісів	Змога запропонувати продукт більшій кількості потенційних користувачів
2.	Поява нових підходів до пошуку оптимального шляху розв’язків	Надання нового функціоналу для надання результатів семантичного компонування	Розробка нового функціоналу з використанням машинного навчання, який би допоміг краще знаходити оптимальні шляхи компонування сервісів
3.	Стрімке зростання росту ринку	Компаніям, що тільки виходять на ринок, буде простіше отримати клієнтів	Змога запропонувати продукт більшій кількості потенційних користувачів
4.	Обслуговування додаткових груп споживачів	Поява нових потенційних груп споживачів	Змога розширити продукт для подальшого впровадження у нові галузі
5.	Розширення асортименту можливих послуг	Поява нового функціоналу, що привабить нових користувачів	Розробка нового функціоналу, що є потребою певної групи користувачів

У Таблиці 5.7 наведено фактори можливостей ринкового впровадження проекту, серед яких: стрімкий ріст попиту на інструменти компонування сервісів , стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг; було також запропоновано можливі реакції компанії. Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (табл. 5.8).

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Вказати тип конкуренції - досконала	Існує 4 фірми-конкурентки на ринку	Врахувати ціни конкурентних компаній на початкових етапах створення бізнесу, реклама (вказати на конкретні переваги перед конкурентами)
2. За рівнем конкурентної боротьби - міжнародний	Дві компанії – з іншої країни, дві – з України	Додати можливість вибору мови ПЗ, щоб легше було у майбутньому вийти на міжнародний ринок
3. За галузевою ознакою - внутрішньогалузева	Конкуренти мають ПЗ, яке використовується лише всередині даної галузі	Створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях
4. Конкуренція за видами товарів: - товарно-видова	Види товарів є однаковими, а саме – програмне забезпечення	Створити ПЗ, враховуючи недоліки конкурентів
5. За характером конкурентних переваг - нецінова	Вдосконалення технології створення ПЗ, щоб собівартість була нижчою	Використання менш дорогих технологій для розробки, ніж використовують конкуренти
6. За інтенсивністю - не марочна	Бренди відсутні	-

У Таблиці 5.8 наведено ступеневий аналіз конкуренції на ринку, де було визначено особливості конкурентного середовища та їх вплив а діяльність підприємства. Однією з найбільш важливих дій компанії для досягнення конкурентоспроможності є необхідність створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (Таблиці 5.9).

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	*Прямі конкуренти в галузі*	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*
	Навести перелік прямих конкурентів	*Визначити бар'єри входження в ринок*	*Визначити фактори сили постачальників*	*Визначити фактори сили споживачів*	*Фактори загроз з боку замінників*
Висновки:	Існує 4 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 3, так як його рішення використовує теж графову базу даних	Так, можливості для входу на ринок є, бо наше рішення передбачає легкий і ефективний підхід до опису семантики сервісів	Постачальники відсутні.	Важливим для користувача є точність роботи компонувальника	Товари-замінники можуть використати більш дешеву технологію створення ПЗ та зменшити собівартість товару

Було здійснено аналіз конкуренції в галузі за М. Портером, в результаті чого було визначено, що існує 4 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 3, так як його рішення також рішення використовує теж графову базу даних, але можливості для входу на ринок є, бо наше рішення передбачає легкий і ефективний підхід до опису семантики сервісів.

За результатами аналізу таблиці робиться висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію. Також робиться висновок щодо характеристик (сильних сторін), які повинен мати проект, щоб бути конкурентоспроможним на ринку. Другий висновок враховується при формулюванні переліку факторів конкурентоспромож ності у п. 3.6. 3.6) На основі аналізу конкуренції, проведеного в п. 3.5 (табл. 5.9), а також із урахуванням характеристик ідеї проекту (табл. 5.2), вимог споживачів до товару (табл. 5.5) та факторів маркетингового середовища (табл. № 5.6-5.7)

визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлюється за табл. 5.10

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1.	Наявність графової бази даних	Дозволяє швидко та якісно масштабувати систему
2.	Можливість створення запитів природною мовою	Зручність для користувача
3.	Наявність REST API	Дозволяє інтегрувати сервіс у складні системи завдяки універсальному API
4.	Хмарне розгортання	Дозволяє звертатись до бази знань як до сервісу
5.	Прогресивний алгоритм для компонування сервісів	Можливість швидко отримати відповідь на бажний запит

У Таблиці 5.10 наведено обґрунтування факторів конкурентоспроможності, серед яких: наявність графової бази даних, можливість створення запитів природною мовою, наявність REST API та хмарне розгортання. Було також наведено обґрунтування цих факторів.

За визначеними факторами конкурентоспроможності (табл.5.10) проводиться аналіз сильних та слабких сторін стартап-проекту (табл. 5.11).

У наступній таблиці наведено проведення аналізу сильних та слабких сторін стартап-проекту, факторами конкурентоспроможності виступили такі: наявність графової бази даних, можливість створення запитів природною мовою, наявність REST API, прогресивний алгоритм для компонування сервісів та хмарне розгортання.

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з нашим підприємством*						
			-3	-2	-1	0	1	2	3
1.	Наявність графової бази даних	15						+	
2.	Можливість створення запитів природною мовою	20				+			
3.	Наявність REST API	10			+				
4.	Хмарне розгортання	5		+					
5.	Прогресивний алгоритм для компонування сервісів	12					+		

Отже, серед сильних сторін проекту можна виділити наступні: наявність графової бази даних, можливість створення запитів природною мовою, наявність REST API, прогресивний алгоритм для компонування сервісів. Серед слабких сторін можна виділити відсутність можливості створення універсального опису сервісів.

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити 103 прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

У наступній таблиці буде проілюстровано SWOT-аналіз стартап-проекту, тобто його слабкі та сильні сторони, можливості та загрози виходу на ринок.

Таблиця 5.12. SWOT- аналіз стартап-проекту

Сильні сторони: наявність графової бази даних, можливість створення запитів природною мовою, наявність REST API, прогресивний алгоритм для компонування сервісів та хмарне розгортання.	Слабкі сторони: можливість зміни тарифів провайдером хмарного розгортання на платні, відсутність можливості створення універсального опису сервісів
Можливості: стрімкий ріст попиту на інструменти компонування семантичних сервісів, можливість запитів природною мовою, стрімке зростання росту ринку, обслуговування додаткових груп споживачів,розширення асортименту можливих послуг	Загрози: конкуренція, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні, надходження на ринок альтернативних продуктів, уповільнення росту ринку

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів.

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Створення додатку з використанням фреймворків Flask, ReactJS	80%	5 місяці
2.	Створення програми на основі без використання будь- яких фреймворків для обробки даних	45%	12 місяців

З означених альтернатив обирається та, для якої: а) отримання ресурсів є більш простим та ймовірним; б) строки реалізації – більш стислими. Тому обираємо альтернативу (створення додатку з використанням фреймворків Flask, ReactJS).

5.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 5.14). Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1.	Дослідницькі центри	Можливість компонувати REST-сервіси на основі семантичних описів	Великий	Існує 4 конкуренти, які надають схожі, але більш вузькі і дорогі рішення.	Наявність REST API, графовох бази даних
2.	Підприємства	Можливість компонувати REST-сервіси на основі семантичних описів	Середній		Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, зручний клієнтський інтерфейс
Які цільові групи обрано: обираємо підприємства та дослідницькі центри					

За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку. Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку. За М. Портером, існують три базові стратегії розвитку, що відрізняються за ступенем охоплення цільового ринку та типом конкурентної переваги, що має бути реалізована на ринку.

Отже, проілюструвати базову стратегію розвитку можна у вигляді Таблиці 5.15

Таблиця 5.15 – Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспромо жні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку*
1.	Створення веб-сервісу, використовуючи фреймворки Flask та ReactJS	Ринкове позиціонування	Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, наявність, зручний клієнтський інтерфейс	Диференціація

Таблиця 5.16 - Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки*
1.	Ні	Так	Буде, а саме: основною задачею є розробка ПЗ з використанням графової бази даних (конкурент 1)	Зайняття конкурентної ніші

Було обрано таку альтернативу розвитку проекту: створення веб-сервісу, використовуючи Flask та ReactJS, адже завдяки цим технологіям можна досягнути ключових конкурентноспроможних позицій кінцевого продукту.

Отже, було визначено базову стратегію конкурентної поведінки як зайняття конкурентної ніші.

Визначимо стратегію позиціонування у Таблиці 5.17, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 5.17 - Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегія розвитку*	*Ключові конкурентоспроможні позиції власного стартап-проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1.	Наявність універсального API, зручне хмарне розгортання, наявність графової бази даних	Диференціація	Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, наявність графової базт даних	Інтеграція, хмарне розгортання, графова база даних

Отже, було вибрано такі асоціації, які мають сформувати комплексну позицію власного проекту:інтеграція (адже завдяки REST API сервіс просто інтегрувати у існуючі системи), хмарне розгортання (оскільки Flask додатки легко розгортаються в хмарах), графова база даних (у системі використовується така, а саме Neo4J).

5.5 Розробка маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 5.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.18 - Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)*
1.	Наявність універсального API	Додаток реалізований у вигляді RESTful сервісу, що надає відповіді у вигляді JSON, що дає змогу користувачам звертатись до сервісу за допомогою стандартних HTTP POST та GET запитів	Перевага в універсальності на можливості інтегрувати сервіс у існуючі системи.
2.	Можливість зручного хмарного розгортання	Можливість розгорнути додаток всюди, де є функціонал розгортання Flask додатків, а така можливість є у більшості хмарних провайдерів	Користувачі мають змогу працювати з системою віддалено у хмарі
3.	Наявність графової бази даних	Швидкість масштабування додатку	Здатність адаптуватися до вимог рингку
4.	Можливість запитів природною мовою	Легкість використання користувачами	Зручність роботи з продуктом

Отже бачимо, що проект має ключові переваги перед конкурентами, які повністю відповідають потребам цільової аудиторії. Додаток реалізований у вигляді RESTful сервісу, що надає відповіді у вигляді JSON, що дає змогу користувачам звертатись до сервісу за допомогою стандартних HTTP POST та GET запитів, а це є досить універсальним способом для подальшої інтеграції сервісу в інші системи.

Далі у Таблиці 5.19 проілюстрована трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання.

Таблиця 5.19 - Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>І. Товар за задумом</td><td colspan="3">Веб-сервіс, що надає можливість оркестрування REST-сервісів на основі семантичного опису, створення запитів природною мовою, хмарного розгортання</td></tr>
<tr><td rowspan="5">ІІ. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Наявність універсального API
2. Можливість зручного хмарного розгортання
3. Наявність графової бази даних,
4. Сприйняття природної мови</td><td>1.Нм
2.Нм
3.Нм
4.Нм</td><td>1.Технологічна
2.Технологічна
3.Технологічна
4.Технологічна</td></tr>
<tr><td colspan="3">Якість: згідно до стандарту ISO 4444 буде проведено тестування</td></tr>
<tr><td colspan="3">Маркування відсутнє</td></tr>
<tr><td colspan="3">Моя компанія: “S.E.M.”</td></tr>
<tr><td rowspan="2">ІІІ. Товар із підкріплюнням</td><td colspan="3">3-місячна пробна безкоштовна версія</td></tr>
<tr><td colspan="3">Постійна підтримка для користувачів</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: патент</td></tr>
</table>

Було описано три рівні моделі товару, з чого можна зробити висновок, що основні властивості товару у реальному виконанні є нематеріальними та технологічними. Також було надано сутність та складові товару у задумці та товару з підкріпленням.

Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. У даному випадку найбільш вірогідним гарантом буде патент.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 5.20). Аналіз проводиться експертним методом.

Таблиця 5.20 - Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники, грн.*	*Рівень цін на товари-аналоги, грн.*	*Рівень доходів цільової групи споживачів, грн.*	*Верхня та нижня межі встановлення ціни на товар/послугу, грн.*
1.	7000	2600	230000	50000-75000

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 5.21).

Таблиця 5.21 - Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1.	Придбання підписки та оплата щомісячних внесків для продовження ліценції	Продаж	0(напряму), 2(через двох посередників)	Власна та через посередників

Отже, система приноситиме прибуток завдяки щомісячним внескам для подовження ліцензії та придбанням підписок, продаж буде виконуватись напряму або через двох посередників.

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 5.22).

Таблиця 5.22 - Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуютьс я цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1.	Придбання ліцензії на користування в мережі Інтернет, щомісячне її продовження, користування сервісом у хмарі або ж на власних серверах.	Інтернет	Інтеграція, хмарне розгортання, графова база даних	Показати переваги сервісу, у тому числі і перед конкурентами	Демо-ролик із використання, рекламні оголошення на популярних сайтах.

Отже, в Таблиці 5.22 наведено концепцію маркетингових комунікацій, було визначено, що придбання ліцензії на користування буде здійснюватись в мережі Інтернет, необхідним буде щомісячне її продовження, користування сервісом можливе у хмарі або ж на власних серверах.

5.6 Висновки

Згідно до проведених досліджень існує можливість ринкової комерціалізації проекту. Також, варто відмітити, що існують перспективи впровадження з огляду на потенційні групи клієнтів, бар’єри входження не є високими, а проект має дві значні переваги перед конкурентами. Для успішного виконання проекту необхідно реалізувати програму із використанням засобів Flask, RecatJS. Для успішного виходу на ринок у продукту повинні бути наступні характеристики:

- наявність універсального API
- можливість зручного хмарного розгортання
- наявність графової бази даних
- можливість створення запитів природною мовою

В рамках даного дослідження були розраховані основні фінансово-економічні показники проекту, а також проведений менеджмент потенційних ризиків. Проаналізувавши отримані результати, можна зробити висновок, що подальша імплементація є доцільною.

Було визначено такі сильні сторони: наявність графової бази даних, можливість створення запитів природною мовою, наявність REST API, можливість хмарного розгортання. Серед слабких сторінможна виділити можливість зміни тарифів провайдером хмарного розгортання на платні, відсутність можливості створення універсальних описів.

Можливості для виходу на ринок включають стрімкий ріст попиту на інструменти компонування REST-сервісів, на основі семантичних описів, пошуку оптимального шляху створення композицій шляхом аналізу вхідних даних стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг. Наявні такі фактори загроз: конкуренція, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні, надходження на ринок альтернативних продуктів, уповільнення росту ринку.

ВИСНОВОК

Одним з основних вкладів даної роботи є розробка метамоделі для опису REST-серівсів її реалізації з використанням JSON. Ця пропозиція призводить до полегшеного підходу який здатний моделювати добре відомі веб-інтерфейси галузевого рівня. Ми розглянули цілу низку різноманітних Web API, щоб знайти ті сервіси, які ближче всього до архітектурного підходу REST, та перевірити наш підхід. Наша реалізація дозволяє працювати з вхідними та вихідними параметрами Web API, що грунтуються на сучасних підходах. Мається на увазі, що деякі параметри є необов'язковими, деякі є обов'язковими, деякі з них присутні в заголовку, інші в тілі запиту, інші в схемі URI, а деякі вимагають певних типів даних, як видно на малюнках 3.5 і 3.6. Однак однією з властивостей, яку ми не підтримуємо, є залежність між відповіддю і конкретними значеннями вхідних параметрів. У деяких сервісів структура відповіді може варіюватися в залежності від значень вхідних параметрів.

Другий внесок - реалізація графу на сонові запропонованої метамоделі. Цей граф дозволив не тільки виявити специфічні сервіси але і автоматично компонувати їх з хорошою продуктивністю. Метамодель може бути реалізована по-різному, для наприклад, як онтологія RDF, або з використанням графового підходу. Значною перевагою запропонованого підходу є можливість інтеграції з існуючими описами веб-сервісів, незалежно від їх формату. Він також передбачає взаємодію з семантичним моделям, таким як існуючі онтології і Linked Data. Семантичний шар дозволяє пов'язувати різні сервіси, грунтуючись на їх призначенні, отже, є можливість зробити сервісне компонуваання, приклад якого наводився на рисунках 4.4, 4.5 і 4.6.

Однак одним з недоліків нашого рішення є те, що опис розділений з самим сервісом, і виступає як додатковий шар. Цей фактор негативно впливає на еволюційну здатність сервісів. Крім того, коли генерується велика кількість

композицій, необхідно реалізувати стратегію ранжирування для того, щоб клієнт міг ефективно використовувати пропоновані композиції. Наприклад, рішення з меншою кількістю кроків може бути кращим, оскільки вони можуть виконуватися швидше. Однак для належного вирішення питання потрібна якісна модель, що відобразить інтереси клієнта .

Зазначимо, що наша пропозиція базується на концепції вхідних і вихідних параметрів сервісу, проте існують інші елементи, які можуть бути потенційно цікавими, а саме статус код чи метадані відповіді.

Результати оркестрування сервісів можуть покращаться з детальнішим описом необхідних концепцій, які часто можуть виявитися надто загальними для специфічних очікувань. Крім того, деякі концепції вважалися еквівалентними (наприклад, ідентифікатори і токени). Як правило, це припущення далеке від істини для реальних додатків. Таким чином, може знадобитися більше специфічних вхідних концепцій, які звузять спектр результатів композиції але будуть більш цінні з точки зору бізнесу.

Відзначимо також розробку стартап-проекту, яку було проведено в 5 розділі нашої роботи. В рамках даного дослідження були розраховані основні фінансово-економічні показники проекту, а також проведений менеджмент потенційних ризиків. Проаналізувавши отримані результати, можна зробити висновок, що подальша імплементація є доцільною.

Було визначено такі сильні сторони: наявність графової бази даних, можливість створення запитів природною мовою, наявність REST API, можливість хмарного розгортання.

Що стосується майбутньої роботи, то планується виправлення недоліків описаних вище. На практиці це може виявитись не таким тривіальним завданням, адже кожен з напрямків вдосконалення існуючого рішення вимагає копіткої експериментальної роботи, що грунтуватиметься на обширній теоретичній базі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Chinnici, R., Moreau, J.-J., Ryman, A., & Weerawarana, S. (2007). Web services description language (wsdl) version 2.0 part 1: Core language. W3C recommendation, 26, 19.
2. Dustdar, S., & Schreiner, W. (2005). A survey on web services composition. International journal of web and grid services, 1(1), 1–30.
3. Fielding, R. (2000). Architectural styles and the design of network-based software architectures (Unpublished doctoral dissertation). University of California, Irvine.
4. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). Hypertext transfer protocol–http/1.1. RFC 2616, June.
5. Gregorio, J. (2007). Do we need WADL. http://bitworking.org/news/193/ Do-we-need-WADL.
6. Gruber, T. R. (1993). A translation approach to portable ontology specifications. Knowledge acquisition, 5(2), 199–220.
7. Hadley, M. J. (2006). Web application description language (WADL). Sun Microsystems, Inc.
8. Kelly, M. (2015). JSON hypertext application language. https://tools.ietf .org/html/draft-kelly-json-hal-02.
9. Kopecky, J., Gomadam, K., & Vitvar, T. (2008). hrests: An html microformat for describing restful web services. In Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT’08. IEEE/WIC/ACM International Conference on (Vol. 1, pp. 619– 625).
10. Kopecky, J., Vitvar, T., Bournez, C., & Farrell, J. (2007). Sawsdl: Semantic annotations for wsdl and xml schema. Internet Computing, IEEE, 11(6), 60–67. Lanthaler, M., & Gutl, C. (2013). Hydra: A Vocabulary for Hypermedia-Driven Web ¨ APIs. LDOW, 996.

11. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., . . . others (2004). OWL-S: Semantic markup for web services. W3C member submission, 22, 2007–04.
12. Pautasso, C. (2009). RESTful Web service composition with BPEL for REST. Data & Knowledge Engineering, 68(9), 851–866.
13. Pautasso, C., & Alonso, G. (2005). Flexible binding for reusable composition of web services. In International conference on software composition (pp. 151–166).
14. Richardson, L., Amundsen, M., & Ruby, S. (2013). Restful web apis. ” O’Reilly Media, Inc.”.
15. Rosenberg, F., Curbera, F., Duftler, M. J., & Khalaf, R. (2008). Composing restful services and collaborative workflows: A lightweight approach. Internet Computing, IEEE, 12(5), 24–31.
16. Sporny, M., Kellogg, G., Lanthaler, M., & W3C RDF Working Group. (2014). JSON-LD1.0: a JSON-based serialization for linked data. W3C Recommendation, 16.
17. Vairetti, C., Alarcon, R., & Bellido, J. (2016). A Semantic Approach for Dynamically ´ Determining Complex Composed Service Behaviour. Journal of Web Engineering, 15(3-4), 310–338.
18. Verborgh, R., Arndt, D., Van Hoecke, S., De Roo, J., Mels, G., Steiner, T., & Gabarro, J. ´ (2015). The Pragmatic Proof: Hypermedia API Composition and Execution. CoRR, abs/1512.07780.
19. Verborgh, R., & De Roo, J. (2015). Drawing Conclusions from Linked Data on the Web: The EYE Reasoner. IEEE Software, 32(3).
20. Verborgh, R., Steiner, T., Van Deursen, D., De Roo, J., Van de Walle, R., & Valles, J. G. ´ (2013). Capturing the functionality of Web services with functional descriptions. Multimedia tools and applications, 64(2), 365–387. Webber, J., Parastatidis, S., & Robinson, I. (2010). Rest in practice: Hypermedia and systems architecture. ” O’Reilly Media, Inc.”.

21. Xie, C., Cai, H., & Jiang, L. (2013). Ontology Combined Structural and Operational Semantics for Resource-Oriented Service Composition. Journal of Universal Computer Science, 19(13), 1963–1985.